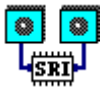
SRI
Small Vision System



User's Manual
Software version 2.1b
February 2001

©Kurt Konolige and David Beymer
SRI International
konolige@ai.sri.com
http://www.ai.sri.com/~konolige

1 Introduction 4

1.1 Implementations of the Stereo Engine 5

1.2 The Small Vision System 6

1.3 Hardware and Software Requirements 7

1.3.1 Analog Framegrabbers 7

1.3.2 Digital Framegrabbers 7

1.4 The SVS Distribution 8

2 Getting started with `smallv` 9

2.1 Inputting Live Stereo Video 10

2.1.1 Stereo heads 10

2.1.2 Analog Framegrabbers 11

2.1.3 IEEE 1394 (FireWire) Framegrabber 12

2.1.4 Selecting Video Format 12

2.1.5 Frame Size 13

2.1.6 Image Sampling 13

2.1.7 Image Source 14

2.1.8 Streaming Mode 14

2.1.9 Adjusting Video Parameters 14

2.1.10 Subwindowing 15

2.1.11 Vergence 16

2.1.12 Color 17

2.2 Storing, Saving, and Loading Stereo Data 18

2.2.1 Stereo Stream Storage 18

2.2.2 Loading and Storing Files 18

2.3 Display 19

2.4 Stereo Processing and Parameters 20

2.4.1 Stereo Function 20

2.4.2 3D Transformation 20

2.4.3 Calibration 20

2.4.4 Disparity Search Range 21

2.4.5 Adjusting the Horopter 22

2.4.6 Pixel Information 22

2.4.7 Correlation Window Size 22

2.5 Filtering 23

2.5.1 Confidence Filter 23

2.5.2 Left/Right Filter 23

2.6 Saving and Restoring Parameters 24

3 Stereo Geometry 26

3.1 Disparity 27

3.2 Horopter 29

3.3 Range Resolution 32

3.4 Area Correlation Window 33

3.5 Filtering 35

3.6 Performance 37

4 Calibration *38*

4.1 Calibration Procedure 39

4.1.1 Calibration procedure steps 39

4.1.2 Imager Characteristics 40

5 API Reference *42*

5.1 Stereo Parameter Structure 43

5.2 Acquiring images from the video capture board 45

5.2.1 Video Capture Interface Libraries 45

5.2.2 Writing a Framegrabber Interface Library 45

5.2.3 Opening the Capture Device 45

5.2.4 Capture Device Format Parameters 46

5.2.5 Frame Size Parameters 46

5.2.6 Digitization Parameters 48

5.2.7 Checking Image Parameters 49

5.3 Continuous Capture 51

5.4 Single Capture 52

5.5 Stereo Engine functions 53

5.5.1 Parameter Structure `svsSP` 53

5.5.2 MMX 53

5.5.3 Warping 53

5.5.4 Stereo Calculation 54

5.5.5 3D Reconstruction 55

5.6 Host display functions 56

5.7 Host file functions 57

1 Introduction

The SRI Stereo Engine is an efficient realization of an area correlation algorithm for computing range from stereo images. Figure 1 shows the results of running the algorithm on a typical scene. The image on the top is the left image of an original stereo pair, while the one on the bottom is a disparity image computed from the stereo pair. In the disparity images, brighter pixels show where the projection of an object diverges between the images (has a high disparity). These are areas that are closer to the cameras. Dark areas have lower disparity, and are further away.

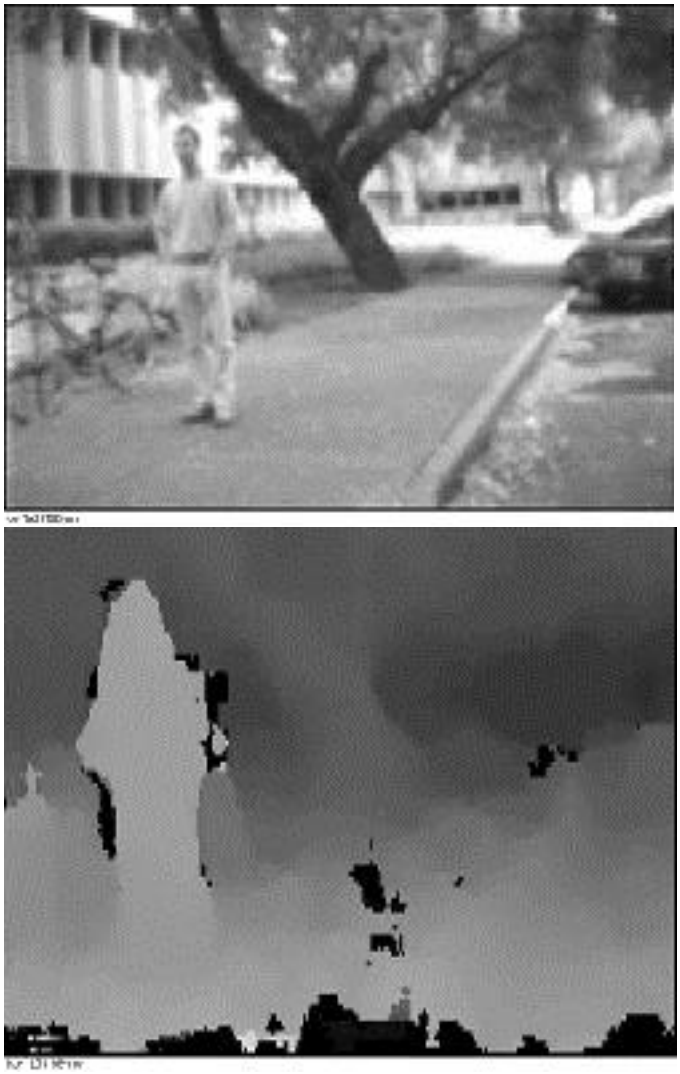


Figure 1-1. An input image and the resultant stereo disparity image. Brighter areas are closer to the camera.

1.1 Implementations of the Stereo Engine

The Stereo Engine exists in several implementations, including embedded, low-power systems and general purpose microcomputers. The embedded systems, or Small Vision Modules (SVMs), contain DSPs or other standalone processors, and produce digital range information. They are meant for end applications where size, cost, and power limitations are critical. SRI will develop embedded SVM systems in partnership with companies who are interested in a particular application.

The Small Vision System (SVS) is an implementation of the Stereo Engine on general-purpose microcomputers, especially PCs running Linux or Windows 95/98/2000/NT. It consists of a set of library functions implementing the stereo algorithms. Users may call these functions to compute stereo results on any images that are available in the PC's memory. Typically, standard cameras and video capture devices are used to input stereo images. The Small Vision System is a development environment for users who wish to explore the possibility of using stereo in an application.

This manual is useful as a source of general information about the Stereo Engine for any implementation, but is also specifically aimed at the development environment of the SVS. It explains the core characteristics of the Stereo Engine, serves as a reference for the stereo function API, and discusses sample applications that use the API. There are also several tutorials that illustrate writing programs to the SVS API, in the documentation folder.

1.2 The Small Vision System

The Small Vision System (SVS) is meant to be an accessible development environment for experimenting with applications for stereo processing. It consists of a library of functions for performing stereo correlation. Figure 1-2 shows the relationship between the SVS library and PC hardware.

Images come in via a pair of aligned video cameras, called a stereo head. A video capture board or boards in the PC digitizes the video streams into main memory. The SVS functions are then invoked, and given a stereo pair as an argument. These functions compute a disparity image, which the user can display or process further.

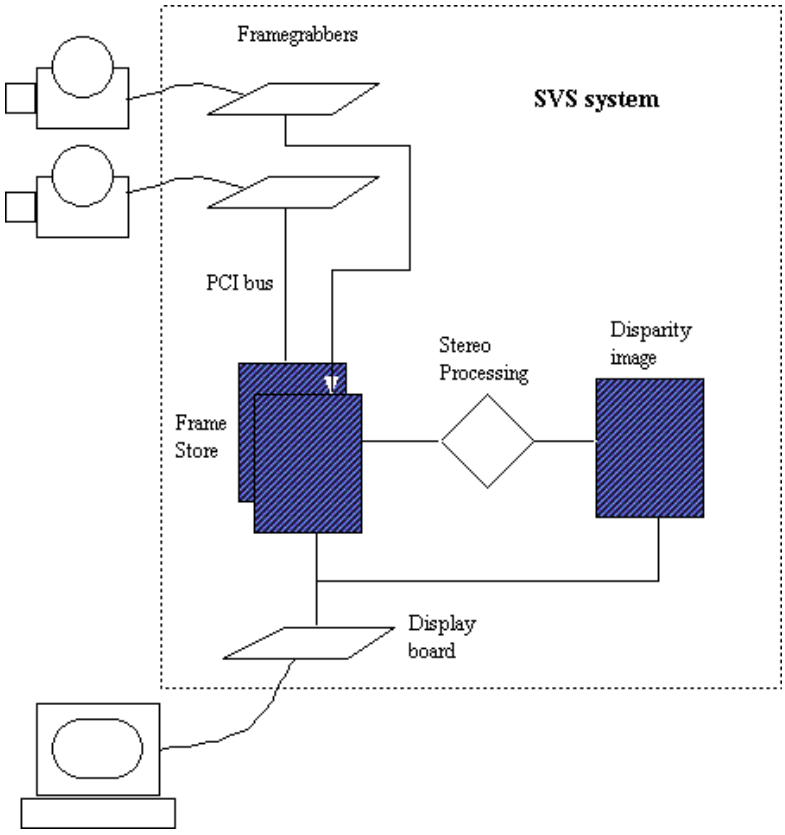


Figure 1-2. The development environment of the Small Vision System.

The SVS environment of Figure 1-2 shows a typical setup for stereo processing of video images. The user may supply his or her own cameras: the SVS has special processing for dealing with camera distortion and calibrating the stereo image (Section 2.4.2). Special stereo heads are also available from from Videre Design (www.videredesign.com). The STH-V3 is an analog head with the ability to send a stereo pair on just a single video signal, so only one video capture device is required (more information on video capture is in Section 2.1). The MEGA-D (STH-MD1) is an all-digital device with megapixel imagers that uses the 1394 bus (FireWire) for direct digital input. Finally, other sources of images may also be used, as long as the images can be placed in PC memory. Some examples are images stored on disk, or images obtained from other devices such as scanning electron microscopes.

1.3 Hardware and Software Requirements

The SVS libraries exist for most Unix systems, as well as MS Windows 95/98/2000/NT; that is, on the most common computer platforms available. We have spent considerable effort in optimizing SVS for PCs using the MMX instruction set (Pentium MMX and Pentium II, III), and it will perform best on these platforms, using either Linux or MS Windows. Performance information is in Section 3.6.

1.3.1 Analog Framegrabbers

Because of the ubiquity of PCs, we have added support for several PCI bus video capture devices on PCs. The following are recommended PC hardware configurations for the SVS with analog cameras.

Operating System	Video Capture Card
Linux	**Imagenation PCX200** **Matrox Meteor and Meteor RGB** **Any Bt848-based card (e.g., Intel Smart Video Recorder III)**
Windows 95/98	**Imagenation PXC200** **Matrox Meteor, Meteor RGB, Meteor II** **Video for Windows device (inferior)** **MRT Video Port Pro (laptop)**
Windows NT, 2000	**Imagenation PXC200** **Matrox Meteor, Meteor RGB, Meteor II**

Table 1-1. Analog Framegrabbers and OS requirements for the Small Vision System.

Unfortunately, as of this time there are no good analog framegrabbers for laptops. The MRT Video Port Pro is one of the best cards, but it is still slow for input to memory, and does not take advantage of the 32-bit CardBus specification. However, the MEGA-D digital device (see below) does have laptop input capability.

1.3.2 Digital Framegrabbers

The MEGA-D (STH-MD1) is an all-digital device that uses the IEEE 1394 (FireWire) bus. Some desktops and laptops have 1394 ports integrated directly into their motherboards. Otherwise, a standard 1394 PCI board or PCMCIA card can be used. The card must be OHCI (Open Host Controller Interface) compliant, which almost all boards are.

1.4 The SVS Distribution

The SVS distribution can reside in any directory; normally, it is placed in `c:\svs` (Windows 95/NT systems) or `/usr/local/svs` or a user's home directory (Unix systems). Here is the directory structure of the SVS distribution.

svs	
readme	installation guide
update	release notes
docs	documentation
smallv.pdf	PDF version of the User Manual
bin	executable and library files
smallv(.exe)	full-featured client demo
stframe(.exe)	simple client example program
stfr(.exe)	simple client example program
stcap(.exe)	frame capture program
XXXgrab.dll,.lib	framegrabber interface fns (Windows OS)
svs.lib	SVS library (Windows OS)
libsvs.so	SVS library (Unix)
fltkdll.dll	Display library (Windows OS)
libfltk.so	Display library (Unix)
data	stereo images
seq-cal.ssi	30 frame sample sequence, calibrated
seq.ssi	30 frame sample sequence, uncalibrated
*-L/R/C.bmp	Stereo pairs and color files
*.ini	Calibration files
samples	sample client program sources
src	SVS library sources
svs.h	main library header
image_io.c	file read and write functions
XXXgrab.c	framegrabber interface functions
libraries	libraries for calibration procedure

2 Getting started with *smallv*

The `smallv` program is a standalone application that exercises the SVS library. It is a GUI interface to the stereo programs, and in addition can load and save stereo image sequences. The `smallv` program is a useful tool for initial development of a stereo application, and can also be used to check out and adjust a stereo camera setup.

The `smallv` program is in the `bin/` directory. It requires shared libraries for the stereo algorithms (`svs`), display (`fltk`), and calibration (various), all of which are in the `bin/` directory. Under MS Windows, these shared libraries (DLLs) must be in the same directory as the `smallv` program, or in the system DLL directory. Under UNIX, the LD_LIBRARY_PATH variable must have the path to the libraries.

Figure 2-1 shows the startup screen of the program. The black windows are for display of image and stereo data. The display programs in SVS use the FLTK cross-platform window interface, and work best in 24 bit mode. The version of the program is indicated in the text information area, and the title bar.

`smallv` will accept stereo images from either a live video source, or a stored file. The easiest way to get started with the program is to open a stored stereo sequence. From the `File` menu, choose `Open`, and navigate to the `data/` directory. The file `seq-cal.ssi` contains 30 stereo frames at 320x240 resolution. When you open it, it begins to display continuously in the display windows. In the `Function` area, pull down the list box and choose `Stereo`. You should see a green pattern representing stereo disparities in the right window. Clicking the `3D Display` button brings up an OpenGL window with a 3D view of the stereo data.

The rest of this section explains the operation of `smallv`. Since `smallv` exercises most of the functionality of the SVS libraries, it should serve as a general introduction to the SVS functions. If you are interested in using a particular framegrabber and set of cameras with `smallv`, please see Section 2.1. The framegrabber interface is indicated in the message area on startup. In this case, it is the MEGA-D digital stereo head.

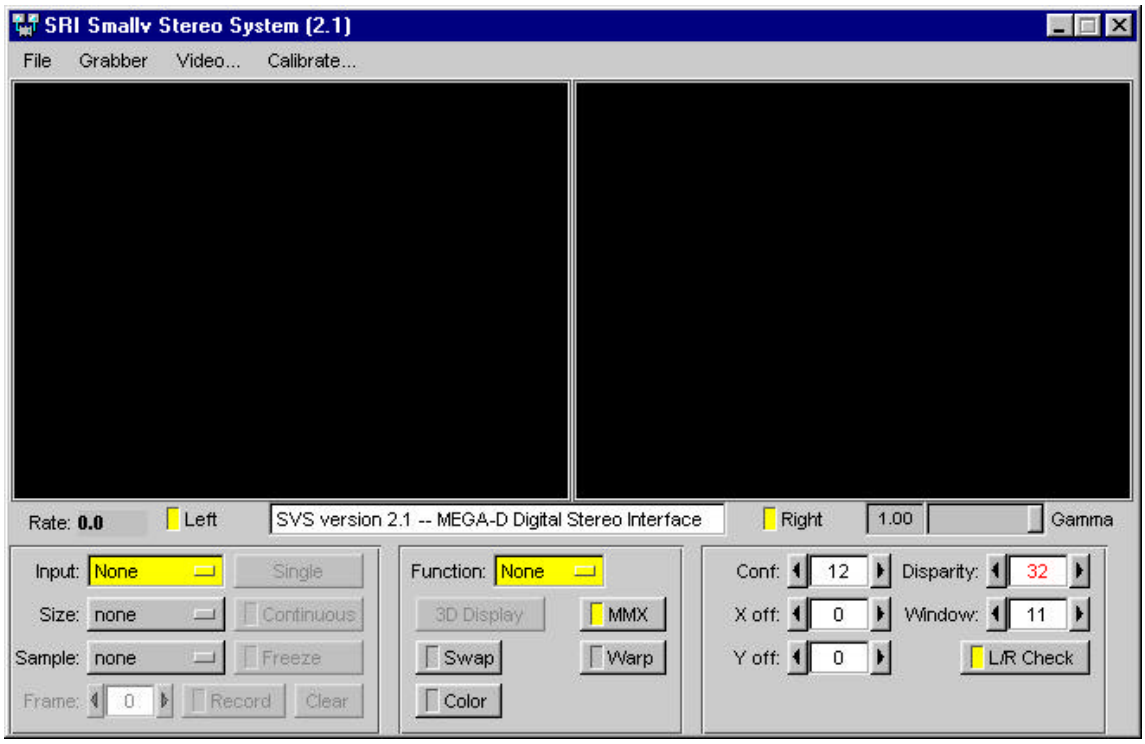


Figure 2-1 Smallv program interface. The two black windows are for display of input images and stereo results.

2.1 Inputting Live Stereo Video

The SVS libraries provide support for live video as stereo input. To input video, you must do the following steps.

1. Decide on a stereo head and framegrabber.
2. Install the framegrabber, following instructions that come with the framegrabber or the stereo head (STH-V3 or MEGA-D).
3. Copy the appropriate framegrabber DLL to `bin\svsgrab.dll` (MS Windows), or `bin/libcap.so` (Unix); see Section 2.1.2).
4. Set the appropriate video format using the Video Format menu.
5. Set the video frame size.
6. Set the input mode to video.

This section gives details necessary for performing these steps.

2.1.1 Stereo heads

Stereo requires two images from different viewpoints. The most common way to get these images is to use two identical cameras separated by a horizontal baseline. It is important the cameras have lenses with the same focal length, and that the pixel elements have the same size.

The baseline is typically from 3 to 8 inches wide, and the cameras are aligned parallel to each other,

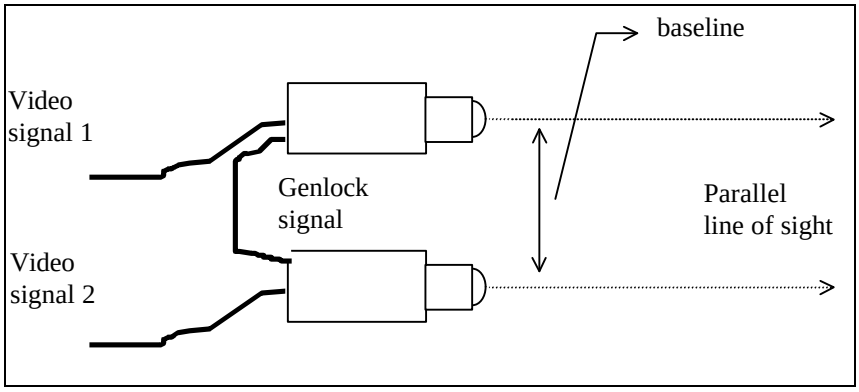


Figure 2-2 Stereo camera setup. Cameras are positioned with parallel lines of sight. Their video signals are synchronized using cross-fed signals.

although other configurations are possible. Figure 2-2 shows a typical stereo camera setup. Two cameras are pointed in the same direction, and they are connected by a cable that *genlocks* the cameras, that is, synchronizes them so that they capture images at the same time. Genlocking is important if there is any motion in the scene. If the cameras are not genlocked, they can capture the image at slightly different times, and any moving objects will be at a slightly different position in one camera relative to the other, than if it they had taken the image at the same time. If the scene is static, then genlocking is not necessary. Not every camera can be genlocked; check that the ones you have can be.

A word about monochrome vs. color cameras. If your application does not need color, it is preferable to use monochrome cameras, because stereo relies only on the luminance component of the video signal. Monochrome cameras have much better spatial resolution and dynamic range than color cameras of the same quality, since they do not have to deal with three color channels.

The cameras produce two video outputs, which must be input to the PC running the SVS system. There are three ways to get these video signals into the PC.

1. Use two framegrabbers, and input one signal on each.
2. Use a single framegrabber capable of inputting 2 monochrome channels, e.g., the Matrox Meteor RGB or Meteor II / Multichannel.

3. Use a single framegrabber, and a stereo head that interlaces two video signals onto a single video stream. The STH-V3 from Videre Design (www.videredesign.com) is one such stereo head.
4. Use a digital stereo head, the MEGA-D (STH-MD1) from Videre Design. This stereo head outputs a digital signal on the 1394 bus, and any OHCI (Open Host Controller Interface) card can be used to input the video.

The SVS libraries can work with any size video frame up to 1288 by 1032 pixels. Standard NTSC cameras capture frames up to 640 by 480, as a set of two fields, each 640 by 240. The camera first captures a field in 1/60 of a second (the *even* field), then captures a second field 1/60 of a second later (the *odd* field). The framegrabber can put these together to form a single image of 640 by 480 size. However, the same problem with motion between non-genlocked cameras can occur on a single camera that combines fields. There is a slight time delay between fields, leading to *motion blur* in the composed frame. For this reason, the SVS libraries use fields rather than frames, so the maximum video size for NTSC signals is 640 by 240.

PAL cameras are also support by the SVS libraries, as long as the framegrabber can input PAL video. They involve similar considerations, but their maximum field size is 768 by 288.

Larger frame sizes with synchronized cameras are possible by using nonstandard *progressive scan* analog cameras, or digital cameras.

2.1.2 Analog Framegrabbers

The SVS libraries include support for a number of popular analog signal framegrabbers, as well as IEEE 1394 digital cards. The table below lists them according to their operating system.

Operating System	Framegrabber	Library
Linux	Matrox Meteor, Meteor RGB, Meteor PPB	`libmet.so`
	Any Bt848-based card, e.g. Intel Smart Video Recorder III Imagenation PXC2000	`libmet.so`
	1394 OHCI card	`libpix.so`
MS Windows 95/98/2000	Matrox Meteor, Meteor RGB, Meteor PPB	`svsmet.dll`
	Matrox Meteor II	`svsmet2.dll`
	Imagenation PXC200	`svspxc.dll`
	MRT VideoPort Pro PC card (single card only, slow)	`svsvpp.dll`
	Any Video for Windows card (single card only)	`svsvfw.dll`
	1394 OHCI card (Windows 98/2000 only)	`svspix.dll`
MS Windows NT 4.0	Matrox Meteor, Meteor RGB, Meteor PPB	`svsmet.dll`
	Meteor II	`svsmet2.dll`
	Imagenation PXC200	`svspxc.dll`

Table 2-1 Framegrabbers supported by SVS.

Under MS Windows 95/NT, a particular framegrabber is accessed from the SVS libraries by copying the corresponding DLL to `svsgrab.dll`, and the library file (`.lib`) to `svsgrab.lib`. For example, if you have installed the Matrox Meteor board, then copy the file `bin/svsmet.dll` to `bin/svsgrab.dll`, and `bin/svsmet.lib` to `bin/svsgrab.lib`. Whenever the SVS libraries are invoked, the default framegrabber will be the PXC200 driver. The default must be set up before starting `smallv`; it cannot be changed while the program is running.

Under Linux, the interface to either the Matrox Meteor cards or a Bt848 card (PXC200, Intel Smart Video Recorder, etc.) is with the shared library `metcap.so`. Copy this library to `bin/libcap.so` to use it. You must load the proper low-level driver for the card; see ftp://ftp.rwii.com/pub/linux/system/Meteor/meteor-1.5.4.tar.gz for Meteor cards, and http://robots.activmedia.com/bt8xx for Bt848 cards. SVS expects the devices to be called /dev/fg0 and

/dev/fg1 (if there are two). Check the device names for your installed devices, and make soft links to the SVS names, e.g., for Matrox Meteor cards, use the command

```
ln -s /dev/mmetgrab0 /dev/fg0
```

The framegrabber interface used by smallv under MS Windows or Linux is indicated in the message window at startup.

There are some limitations in framegrabber drivers that should be noted. First, there are currently no fast framegrabbers for portables, although this could change if they take advantage of the PC card 32-bit interface. The maximum frame rate for the VideoPort Pro, one of the better cards, is about 15 fps at 320 x 120; this frame rate uses all of the CPU to do the I/O transfers.

The Video for Windows interface has limitations that prevent it from working in Windows NT. Even in Windows 95, it is inefficient, since there is no way to specify field mode captures, and the fields are extracted by SVS after a full frame has been shipped to memory. We recommend using a Matrox or Imagenation card under MS Windows.

When using the Imagenation PXC200 cards with PC104+ form factor under Linux, an additional initialization call is necessary. The variable `svsIsPXC_PC104` should be set to 1, which will be done automatically if the environment variable `SVS_PXC_PC104` is defined.

2.1.3 IEEE 1394 (FireWire) Framegrabber

The SVS has an interface to digital stereo heads from Videre Design via the IEEE 1394 serial bus. Any OHCI-compliant IEEE 1394 PCI or PCMCIA card can be used, under MS Windows 98/2000 or Linux. Please check the stereo head manual for instructions on installing the 1394 card and drivers. The relevant DLL for MS Windows is `svspix.dll`, and the shared library for Linux is `libpix.so`.

2.1.4 Selecting Video Format

The Video Format menu lets you tell the SVS library what kind of video input you are using. This section should be ignored if you are using the IEEE 1394 digital interface with a Videre Design digital head, which has a fixed format.

There are three choices:

1. Line interlace [default]. This is the mode for the STH-V3 line interlace stereo head. Any framegrabber can be used in this mode.
2. Dual framegrabbers. This mode uses two framegrabbers, with one framegrabber per camera. Check Table 2-1 for supported framegrabbers.
3. RG components. This mode uses the Matrox Meteor RGB or Meteor II, and inputs one camera video stream on the R channel, and one on the G channel.

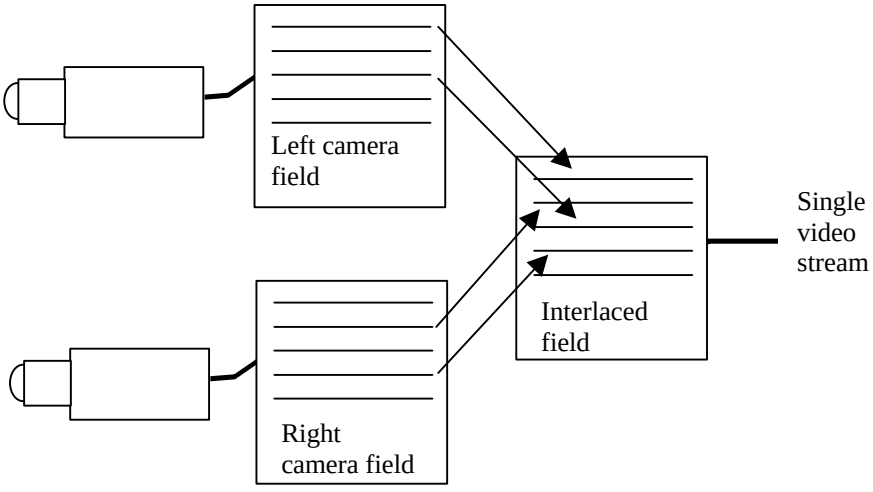


Figure 2-3 Line-interlaced stereo cameras (STH-V1 stereo head). Alternate lines from each camera are interlaced into a single video stream.

It is important for stereo processing to have the left camera image appear as the left image in the `smallv` program. Once the video input is displayed, you can check this by pointing the cameras along your line of sight. The right camera appears on your right side, and the right image on the `smallv` display should show this image. You can cover one camera with your hand, and observe which displayed image goes dark. With dual framegrabbers or RG input, the solution to having the wrong camera inputs is to simply switch the inputs, or to use the `swap` button in `smallv`, which interchanges the images in memory.

Under line interlace mode (Figure 2-3), the first horizontal line of a video field is from the left camera, the second from the right, the third from the left, and so on, making a single video stream. The SVS software de-interlaces the video stream, reconstructing the left and right images in memory, at half the original vertical resolution. Because of the variation in how framegrabbers determine which is the first line of a field, the SVS software will sometimes switch the left and right fields during de-interlacing. The `swap` button switches the left and right fields during deinterlacing.

2.1.5 Frame Size

The SVS libraries as delivered can work with frame sizes up to 1288 by 1032. In fact, the SVS algorithms can work with arbitrarily sized frames.

A subset of frame sizes are supported for video input in `smallv`; the following table summarizes them. Most framegrabbers support hardware interpolation and scaling, so that bus traffic is minimized by working with smaller frames. The exception is the Meteor RGB, which passes a full field to memory, where it is decimated by the SVS software.

Video Format	Frame Sizes
Line interlace	160x120
	320x120
Dual framegrabber and RG component	640x240
	320x240
	320x120
	160x120
1394 (digital) interface	1280x960, 640x480, all others above

Table 2-2 Frame sizes available for video input in smallv.

Video frame size is selected with the `Size` drop list in the `Source` area. Video sizes can be changed at any time with most framegrabbers; however, the Video for Windows interface is unstable and may crash if frame sizes are changed during acquisition. It is always best to stop video acquisition before changing frame sizes.

2.1.6 Image Sampling

The sampling for analog framegrabbers is implicit in the frame size. For example, if the camera image size is 320x240, and the requested frame size is 160x120, then the full image is scaled down by the framegrabber, usually using interpolation to produce a smooth image.

With the MEGA-D digital interface, the user has full control over the sampling method, and the `Sample` and `Size` controls combine to produce the final result. For example, if the sampling mode is x1 (no subsampling), then and image size of 320x240 produces a *subwindow* within the full image. (Subwindowing is not available from analog framegrabbers supported by SVS.) The placement of the subwindow can be changed in real time under program control, using the dialog from the `video…` menu.

There are several sampling modes. *Decimation* samples the image by removing pixels, e.g., "`x2 dec`" means that every second pixel in a line is removed, and every other line is removed. *Binning* samples the image by averaging over a block of four pixels, to produce the same result. Binning produces smoother images with less noise, but it is slower than decimation, which is done by the stereo hardware.

Combination sampling modes are available, e.g., "`x4 bin+dec`" samples the image down to ¼ size in horizontal and vertical directions, by decimating by 2 and then binning by 2.

2.1.7 Image Source

The source for stereo images can be either a memory buffer or a live video stream. The `Source` drop list lets you choose between these, or to stop any input. Buffer input is discussed in Section 2.1.9.

2.1.8 Streaming Mode

Images from video cameras or the buffer can be processed in three acquisition modes. Only one acquisition mode is active at a given time.

- Continuous mode. In this mode, stereo pairs are continuously input, processed, and displayed. The maximum frame rate is 30 Hz for live analog image data, and up to 80 Hz for the MEGA-D digital system. See Section 3.6 for performance information. The rate is indicated next to the text information area.
- Single frame mode. In this mode, a single stereo pair is input, processed, and displayed each time the Single button is pressed.
- Freeze mode. In this mode, a single stereo pair is input, then the same frame is continuously processed and displayed. This mode is useful in checking the effect of different stereo parameters on the same image.

2.1.9 Adjusting Video Parameters

Most framegrabbers support some kind of video image adjustments, such as contrast or brightness. The video parameter dialog is invoked using the `Video...` menu item (Figure 2-4).

Most analog cameras have automatic adjustment of exposure and gain, which change according to lighting conditions. The user can set brightness and contrast, which are framegrabber parameters that change the processing of the analog signal.

The MEGA-D digital stereo head has manually controlled exposure and gain. Exposure is the time that any given pixel is exposed to light before being read out. Gain is a amplification of the signal that comes out of the pixel. In general, it is best to increase the exposure first, and if necessary, to increase gain once exposure reaches a maximum. The reason for this is that gain will increase the video noise, while exposure increases the pixel's response to light. In some cases, though, short exposure times are

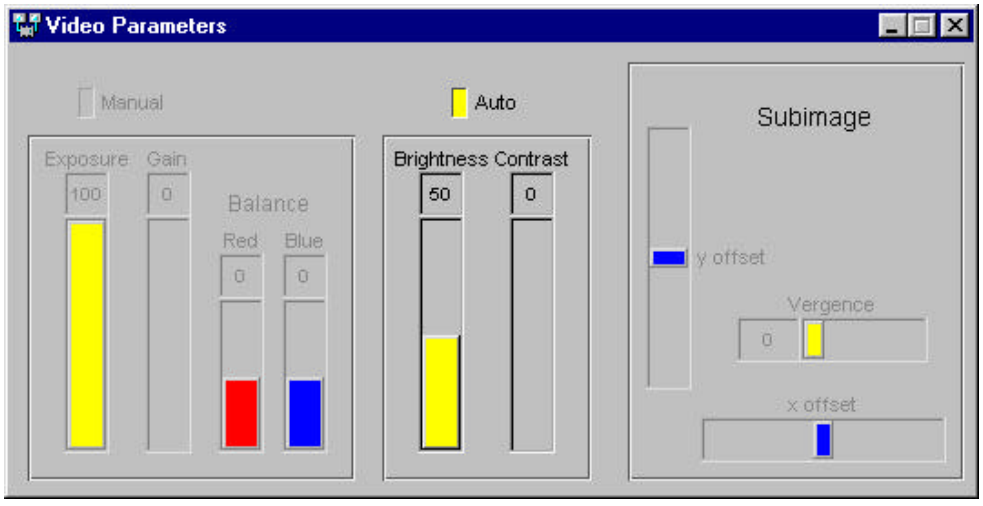


Figure 2-4 Video Parameter dialog box.

Figure 2-5 Two 320x240 subwindows (bottom) of a 640x480 image (top).

desirable for minimizing motion blur, and it may be more convenient to increase gain while exposure is not at a maximum.

The values of exposure, gain, brightness, and contrast are all represented as a percent.

The colorized version of the MEGA-D digital camera can input color images, and the color balance can be adjusted manually using the red/blue differential gain. More information about color processing is in Section 2.1.12.

2.1.10 Subwindowing

The MEGA-D digital stereo head can send to the host computer just a portion, or *subwindow*, of the stereo image. For example, if the MEGA-D is in x2 sampling mode (full-size image is 640x320), and the image size is chosen to be 320x240, then `smallv` will input only a 320x240 subwindow of the full image. Figure 2-5 shows two of these subwindows, and the original full-size image.

The placement of subwindows is controlled by the vertical (Y) and horizontal (X) offset controls in the Subimage box of the Video Parameters dialog window. These parameters can be changed in real time, enabling electronic panning of the live image.

2.1.11 Vergence

When in subwindow mode, the two cameras in a stereo rig generally will have the same X and Y offsets, so that they keep the parallel line-of-sight characteristic of the stereo rig. However, for viewing close objects, it is advantageous to toe-in, or *verge*, the two stereo cameras. In this way, the images of the near object will both contain the object in the center.

Human eyes verge mechanically when viewing close objects. Mechanical vergence for stereo cameras is difficult, however, since it involves complicated motor control, and more importantly, disturbs the calibration that is critical for stereo analysis. Instead, with the subwindow capability of the MEGA-D, it is possible to verge the stereo images electronically, by choosing appropriate horizontal offsets for each image.

Figure 2-6 shows the effects of using electronic vergence. The top stereo pair, of a close object, puts the object into the center of the left frame. In the right frame, the object has a large disparity and is visible in the left side of the frame.

The bottom stereo pair is created by adding vergence to the subwindow process, offsetting the right subwindow horizontally by 120 pixels, relative to the left subwindow. Both frames now have the near object centered.

Vergence of the subwindows is set using the vergence control in the `Subimage` box of the `Video Parameters` dialog. It is a real time control, just like the X and Y subwindow offsets.

Figure 2-6 Parallel image subwindows (top) and verged image subwindows (bottom), showing a close object.

2.1.12 Color

As of Version 2.1, SVS supports color input and display. Besides the two monochrome left/right stereo channels, there is a third color channel, with images in RGB 24-bit format. The color channel does not participate in stereo processing, but it can be useful in applications that combine color and stereo information, for example, object tracking.

Color information from the MEGA-D digital head (STH-MD1-C) is input as raw colorized pixels, and converted by the interface library into two monochrome and one RGB color channel. The color channel corresponds to the left image, which is the reference image for stereo. The color image can be de-warped, just like the monochrome image, to take into account lens distortion (Section 4).

Color information from the camera is input only if the `Color` button is pressed on the main window (Figure 2-1).

Because the typical color camera uses a colorizing filter on top of its pixels, the color information is sampled at a lower resolution than a similar non-colorized camera samples monochrome information. In general, a color camera has about ¼ the spatial resolution of a similar monochrome camera. To compensate for the reduced resolution, use binning (Section 2.1.6) to increase the fidelity of the image. For example, if you need a 320x240 frame size, use 640x480 and binning x2.

The relative amounts of the three colors, red/green/blue, affects the appearance of the color image. Many color CCD imagers have attached processors that automatically balance the offsets among these colors, to produce an image that is overall neutral (called *white balance*). The MEGA-D provides manual color balance by allowing variable gain on the red and blue pixels, relative to the green pixels. Manual balance is useful in many machine vision applications, because automatic white balance continuously changes the relative amount of color in the image.

The manual gain on red and blue pixels is adjusted using the `Video Parameters` window (Section 2.1.9). For a particular lighting source, try adjusting the gains until a white area in the scene looks white, without any color bias.

2.2 Storing, Saving, and Loading Stereo Data

`smallv` provides a basic facility for loading and saving stereo data streams. The file load and store functions are part of the SVS library, and their source code is included. `smallv` exercises these functions, and provides a memory buffer for storing live stereo video. In `smallv`, the buffer always holds the left and right input stereo images.

2.2.1 Stereo Stream Storage

`smallv` has an internal buffer capable of holding 30 stereo pairs (frames) at a 640x480 frame size. The buffer will hold more frames at smaller sizes, fewer frames at larger sizes. The buffer can be filled from a previously-saved file, or from live video input. The buffer can also be written out to a file, and used as the source for stereo processing in `smallv`. The current frame is indicated in input information area.

When the input source is the buffer, the acquisition mode controls (`Continuous`, `Single`, `Freeze`) control the processing of the buffer frames (Section 2.1.8). The frame control can also be used to go to an individual frame when in `Single` acquisition mode.

The `Record` button controls the input of live video into the buffer. `Clear` clears the buffer and resets it to frame 0. Activating the `Record` button starts the input of live video frames into the buffer. The source must be set to `Video`; either `Continuous` or `Single` mode may be used. Frames are stored sequentially until the buffer is full. Pressing `Record` again will also turn off acquisition..

As an example, to capture a short video sequence and replay it, perform the following steps.

1. Start acquiring live video in continuous mode.
2. Clear the buffer (`Clear` button).
3. Start buffer storage (`Record` button).
4. After a short period, stop buffer storage (`Record` button).
5. Change from `Video` to `Buffer` source.

At this point, the short segment that is in the buffer will be replayed as a short continuous loop. The buffer, or individual images, can be saved to a file.

The buffer does not save color information, although in future versions of SVS we plan to add this capability.

2.2.2 Loading and Storing Files

The SVS libraries work with two different file types for image storage.

- BMP format. Each BMP file contains a single 8-bit grayscale image, or an RGB 24-bit color image. The color coding for the 8-bit BMP file is 256 shades of gray, with 0 being black and 255 white. By convention, a stereo pair is saved as two files with the linked names `XXX-L`.BMP (left image) and XXX-R.BMP (right image). The corresponding color file is saved as XXX-C.BMP. Stereo disparity information is saved as an 8-bit BMP file XXX-S.BMP.
- SSI format. This is an internal SRI format for storing image sequences. An arbitrary number of stereo pairs are saved in a single file, with header information describing the image size and number of images. Images are 8-bit grayscale; there is no color information.

Images and image sequences are loaded into and stored from the buffer using the `File` menu. To load stereo frames, use the `Load` menu item to bring up a file choice dialog. You may pick either SSI or BMP files to load. Choosing either BMP file of a pair automatically loads the other. In addition, if a color file is present, it is also loaded.

To save the buffer to a file, use the `Store Buffer` menu item. This saves the buffer as an SSI file. Alternatively, to save the current frame as two BMP files, use the `Store Current` menu item. If stereo processing is active, then the stereo disparity image is saved also as a BMP file; this is the only method to save a disparity image from the Smallv application. Color information, if present, is saved as a 24-bit BMP file.

2.3 Display

`smallv` displays two images in its display area. The left display is always the left input image. Input images are displayed in grayscale, unless color information is present: in this case, the left image will be shown in color.

The right display can be either the right input image, or the results of processing. Processing results are always displayed in "greenscale", using shades of green.

Either display can be turned off by unchecking the box underneath the display area. Turning off the display will let `smallv` run faster.

Images larger than 320x240 are automatically scaled down by factors of 2^n to fit into a 320x240 area.

To display properly for human viewing, most video images are formatted to have a nonlinear relationship between the intensity of light at a pixel and the value of the video signal. The nonlinear function compensates for loss of definition in low light areas. Typically the function is x^γ, where γ is 0.45, and the signal is called "gamma corrected." Digital cameras, such as the MEGA-D, do not necessarily have gamma correction. This is not a problem for stereo processing, but does cause the display to look very dark in low-light areas. You can add gamma correction to the displayed image by choosing an appropriate gamma value in the slider under the right display window (Figure 2-1).

2.4 Stereo Processing and Parameters

In `smallv`, stereo processing takes place in conjunction with the input of stereo images. The basic cycle is:

get stereo pair -> process pair -> display pair

The input is either from live video or the buffer (Sections 2.1 and 2.1.9). In freeze mode, the same pair is processed continuously, so adjustments can be made in stereo parameters.

2.4.1 Stereo Function

Stereo processing is turned on by choosing `Stereo` from the `Function` drop list. The stereo disparity image will appear in the right display. Stereo disparities are encoded by green: brighter green is a higher disparity, and therefore closer to the cameras (see Section 2.4.4 for a technical description of disparity).

Disparities represent the distance between the horizontal appearance of an object in the stereo images. The stereo process interpolates this distance to 1/16 pixel, e.g., a disparity value of 45 represents a displacement of 2 13/16 pixels. The maximum displacement currently supported is 80 pixels, so disparity values range from 0 (no disparity) to 1280. Disparity values are returned as 16-bit (short) integers. The values 0xFFFF and 0xFFFE are reserved for filtering results (Section 2.5)

If `smallv` is running on a Pentium MMX processor (MMX, PII, or PIII) then stereo processing is much faster, taking advantage of the parallel data operations. The processor is queried and the MMX box is checked if the instructions are available. You can turn the MMX processing on and off by toggling the box. But, if your system does not have MMX instructions, you will not be able to turn it on.

2.4.2 3D Transformation

A pixel in the disparity image represents range to an object. This range, together with the position of the pixel in the image, determines the 3D position of the object relative to the stereo rig. SVS contains a function to convert disparity values to 3D points. These points can then be displayed in a 3D viewer.

To take the current disparity image and display it in 3D, press the `3D Display` button. An OpenGL window will show the 3D points constructed from the disparity image, and you can change the viewpoint of the window to see the 3D structure.

2.4.3 Calibration

For good stereo processing, the two images must be aligned correctly with respect to each other. The process of aligning images is called *calibration*. Generally speaking, there are two parts to calibration: *internal calibration*, dealing with the properties of the individual cameras and especially lens distortion; and *external calibration*, the spatial relationship of the cameras to each other. Both internal and external calibration are performed by an automatic calibration procedure described in Section 4. The procedure needs to be performed when lenses are changed, or the cameras are moved with respect to each other.

From the internal and external parameters, the calibration procedure computes an image warp for rectifying the left and right images. In stereo rectification, the images are effectively rotated about their centers of projection to establish the ideal stereo setup: two cameras with parallel optical axes and horizontal epipolar lines (see Fig. 2-2). Having the epipolar lines horizontal is crucial for correspondence finding in stereo, as stereo looks for matches along horizontal scanlines.

Figure 2-7 shows a pair of images of the calibration target taken with the MEGA-D stereo head and a 4.8 mm wide-angle lens. In the original images on the top, there is lens distortion, especially at the edges of the image: notice the curve in the target. Also, the images are not aligned vertically.

The bottom pair is the result of calibrating the stereo head and then rectifying the two original images. Now the images are aligned vertically, and all scene lines are straight in the images.

Figure 2-8 shows sample disparity images for uncalibrated and calibrated cameras. Without calibration, it is impossible for the stereo algorithms to find good matches.

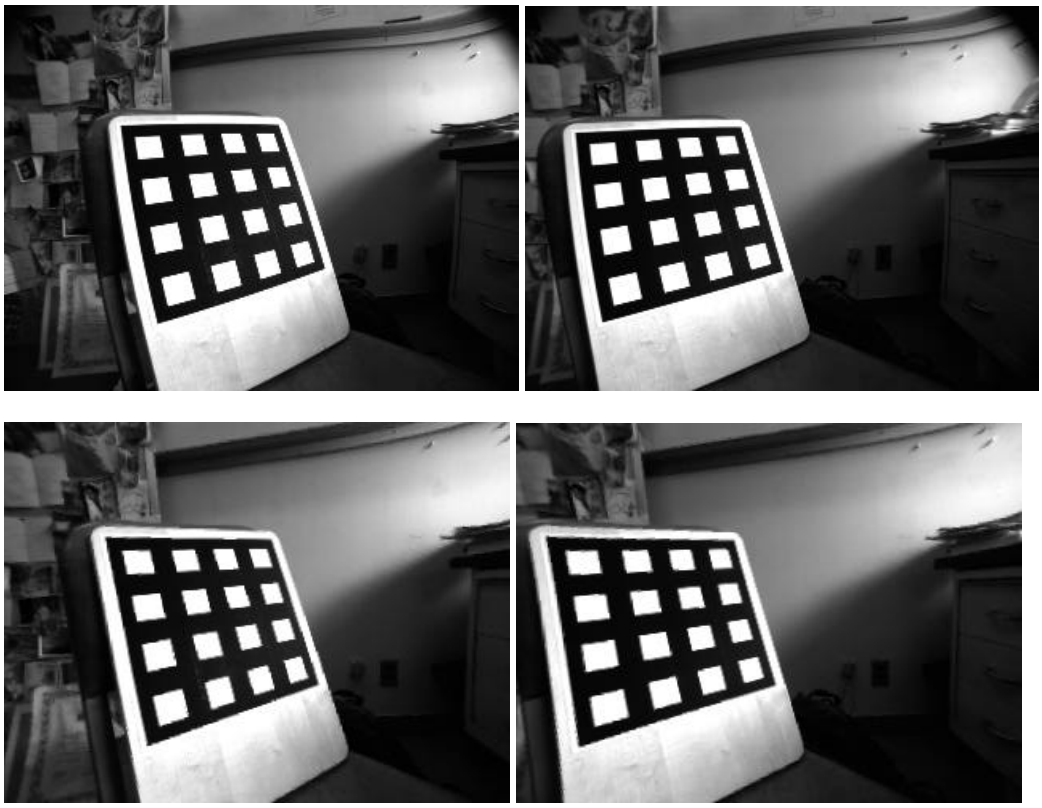


Figure 2-7 Original stereo pair (top) and rectified pair (bottom).

2.4.4 Disparity Search Range

Even with stereo rectification, it may not be possible to match every object in the scene, because the horopter is not large enough. In this case, the horopter can be enlarged by changing the number of disparities searched by the stereo process. This search range can vary from 8 to 80 pixels. Larger search ranges enlarge the horopter, but not in a linear fashion, i.e., a search range of 32 does *not* give twice the horopter range of 16; see Section 4 for technical details.

Changing the disparity search size affects the time it takes to process stereo. A search space of 32 pixels will take about twice as long as a search space of 16 pixels. It will actually take a little less,

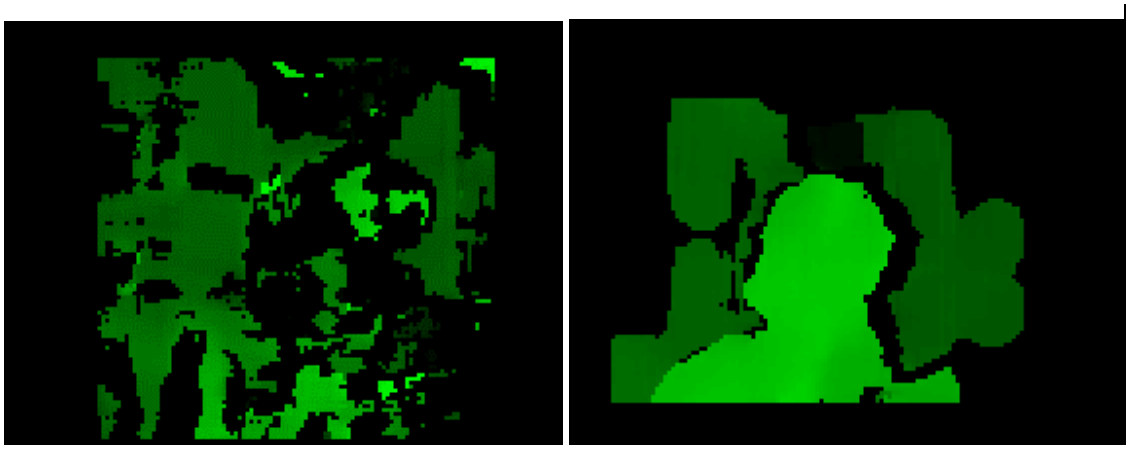


Figure 2-8 Uncalibrated (left) and calibrated (right) disparity images.

because there is some fixed overhead in processing the images. Obviously, the smallest search range necessary for the application is the best choice.

Disparities are interpolated to 1/16 pixel, so a search range of 16 means that there are 256 integral disparity values, ranging from 0 (no disparity) to 255 (maximum disparity of 15 15/16 pixels).

The search range is selected using the `Disparities` drop list in the `Parameters` area. When the range is switched, the disparity image will lighten or darken to reflect the changed values of disparities.

2.4.5 Adjusting the Horopter

The stereo rectification procedure sets up the horopter, or depth of field of stereo, so that objects are matched from infinity to some distance in front of the camera. Objects closer than this near point will not be matched, and will produce random disparity readings. The near point distance is a function of the search size, the stereo baseline, and the focal length of the camera lenses. One can adjust the horopter by adjusting a horizontal X offset, moving the depth range closer to the camera. The depth range desired in the end application would drive the setting of this parameter. For example, if the image does not contain any objects farther than a certain distance, the X offset can be adjusted so that the far point of the horopter is at that distance. Changing the X offset causes the disparity display to get uniformly lighter or darker, as the horopter is shifted and the disparity of an object changes. Adjusting the horopter to cover a specific range of depths is discussed in Section 4.

2.4.6 Pixel Information

SVS will show pixel information when the left button is clicked in either SVS display window. The information is displayed in the text window in the format:

```
x232 y120 [131] [11] Xaaa Ybbb Zccc
```

The image coordinates of the mouse are given by the `x,y` values. The values in square brackets are the pixel values of the left and right images. If the right image is displaying stereo disparities, then the right value is the disparity value. Finally, the `X,Y,Z` values are the real-world coordinates of the image point, in mm. Note that `X,Y,Z` values are calculated only if stereo is being computed, and to be accurate, a good calibration file must be input (Section 4).

2.4.7 Correlation Window Size

A final parameter that affects stereo processing is the size of the correlation window used for matching. A larger window will produce smoother disparity images, but will tend to "smear" objects, and will miss smaller objects. A smaller window will give more spatial detail, but will tend to be noisy. Typical sizes for the window are 9x9 or 11x11. The window size is selected using the `Sum window` drop list. In the MMX implementation, not all window sizes are supported. More technical information on the correlation window can be found in Section 3.4.

2.5 Filtering

Stereo processing will generally contain incorrect matches. There are two major sources for these errors: lack of sufficient image texture for a good match, and ambiguity in matching when the correlation window straddles a depth boundary in the image. The SVS stereo processing has two filters to identify these mismatches: a confidence measure for textureless areas, and a left/right check for depth boundaries.

Areas that are filtered appear black in the displayed disparity image. To distinguish them from valid disparity values, they have the special values 0xFFFF (confidence rejection) and 0xFFFE (left/right rejection).

2.5.1 Confidence Filter

The confidence filter eliminates stereo matches that have a low probability of success because of lack of image texture. There is a threshold, the confidence threshold, that acts as a cutoff. Weak textures give a confidence measure below the threshold, and are eliminated by the filter.

The confidence threshold is adjusted using the `Conf` spin control in the `Parameters` area. A good value can be found by pointing the stereo cameras at a textureless surface such as a blank wall, and starting the stereo process. There will be a lot of noise in the disparity display if the confidence threshold is set to 0. Adjust the threshold until the noise just disappears, and is replaced by a black area.

The computational cost of the confidence filter is negligible, and it is usually active in a stereo application.

2.5.2 Left/Right Filter

Each stereo camera has a slightly different view of the scene, and at the boundaries of an object there will be an area that can be viewed by one camera but not the other. Such occluded areas cause problems for stereo matches. Fortunately, they can be detected by a consistency check in which matching is done first by using the left image as a fixed base, and then repeating the match using the right image as the base. Disparity values for the same point that are not the same fail the left/right check. Typically, this will occur near the boundaries of objects.

The left/right check is controlled by three radio buttons in the `Parameter` area. It can be turned on or off. A third option is to perform the check, but instead of discarding disparity values that are inconsistent, use the one that is smaller (further away). This option can fill in the areas around object borders in a reasonable way. It is not currently available under MMX processing.

The left/right check adds about 20% to the computational cost of the stereo process, but is usually worth the effort.

2.6 Saving and Restoring Parameters

All of the parameters that control the operation of the SVS Stereo Engine can be saved to a file for later use. Parameter files can be loaded and saved using the File menu: Load Param File and Store Param File.

The file `data/megad-75.ini` contains a sample file for a 7.5 mm lens on the MEGA-D stereo rig. It serves as an example of the settings available through parameter files. In practice, these settings are usually computed using the calibration program, and then saved to a file for later use. But, it is also possible to change the settings directly in the file.

```
# SVS Engine v 2.1 Stereo Camera Parameter File

[image]              # frame size and subwindow
max_linelen 1288     # maximum width of camera, in pixels
max_lines   1032     # maximum height of camera, in pixels
max_decimation 4     # 1, 2 or 4 for max decimation capability
max_binning    2     # 1 or 1 for max binning capability
ix 0                 # subwindow offset
iy 0
width 320            # subwindow size
height 240
linelen 320          # full frame size
lines 240
gamma 0.7            # gamma value for display
color 1              # 0 for monochrome, 1 for color

autogain   0         # 1 if has auto gain capability
manualgain 1         # 1 if has manual gain capability
autowhite  0         # 1 if has auto white balance capability
manualwhite 1        # 1 if has manual white balance capability

gain       0         # current gain, -1 if auto
exposure 100         # current exposure, -1 if auto
contrast  50         # current contrast, -1 if auto
brightness 50        # current brightness, -1 if auto

saturation 50        # current saturation, -1 if auto white balance
red         0        # red gain offset, from -40 to 40
blue        0        # blue gain offset, from -40 to 40


[stereo]             # stereo processing parameters
convx 9              # prefilter kernel size
convy 5
corrxsize 11         # correlation window size
corrysize 11
thresh 20            # confidence threshold value
lr 1                 # left/right filter on (1) or off (0)
ndisp 24             # number of disparities to search
dpp 16               # subpixel interpolation
offx 0               # horopter offset
offy 0               # vertical image offset, not used


[external]
Tx -89.458214        # translation between left and right cameras
Ty -0.277252
Tz -0.923279
```

```
Rx -0.008051          # rotation between left and right cameras
Ry -0.003771
Rz -0.000458


[left camera]
pwidth 1280           # number of pixels in the camera
pheight 960
dpx 0.007500          # pixel spacing, mm
dpy 0.007500
sx 1.000000           # aspect ratio
Cx 582.260123         # camera center, pixels
Cy 506.081223
f 7.798704            # focal length, mm
kappa1 0.002983       # radial distortion parameters
kappa2 -0.000040
proj                  # projection matrix: from left camera 3D coords
                      #    to left rectified coordinates
  1.041674e+003 6.177793e+000 5.666963e+002 0.000000e+000
  -6.957139e+000 1.042596e+003 5.022628e+002 0.000000e+000
  -6.576823e-003 -3.900478e-005 1.000000e+000 0.000000e+000
rect                  # rectification matrix for left camera
  1.001883e+000 -5.935817e-003 1.956143e+001
  6.693463e-003 1.000873e+000 1.397722e+000
  6.313708e-006 1.844243e-015 1.000000e+000


[right camera]
pwidth 1280           # number of pixels in the camera
pheight 960
dpx 0.007500          # pixel spacing, mm
dpy 0.007500
sx 1.000000           # aspect ratio
Cx 548.992956         # camera center, pixels
Cy 495.924832
f 7.834438            # focal length, mm
kappa1 0.002722       # radial distortion parameters
kappa2 -0.000021
proj                  # projection matrix: from right camera 3D coords
                      #    to right rectified coordinates
  1.041674e+003 6.177795e+000 5.666964e+002 -9.352453e+004
  -6.957140e+000 1.042596e+003 5.022628e+002 4.191134e-004
  -6.576824e-003 -3.900531e-005 1.000000e+000 -9.225858e-005
rect                  # rectification matrix for right camera
  1.006349e+000 -9.720995e-003 -1.942293e+001
  8.006440e-003 9.997475e-001 -1.474720e+000
  9.887915e-006 -7.757015e-006 1.000000e+000
```

3 Stereo Geometry

Stereo algorithms compute range information to objects by using triangulation. Two images at different viewpoints see the object at different positions: the image difference is called *disparity*. This section discusses the basic equations that govern the relationship between disparity and range.

3.1 Disparity

The figure below displays stereo geometry. Two images of the same object are taken from different viewpoints. The distance between the viewpoints is called the *baseline* (**b**). The focal length of the lenses is **f**. The horizontal distance from the image center to the object image is **dl** for the left image, and **dr** for the right image.

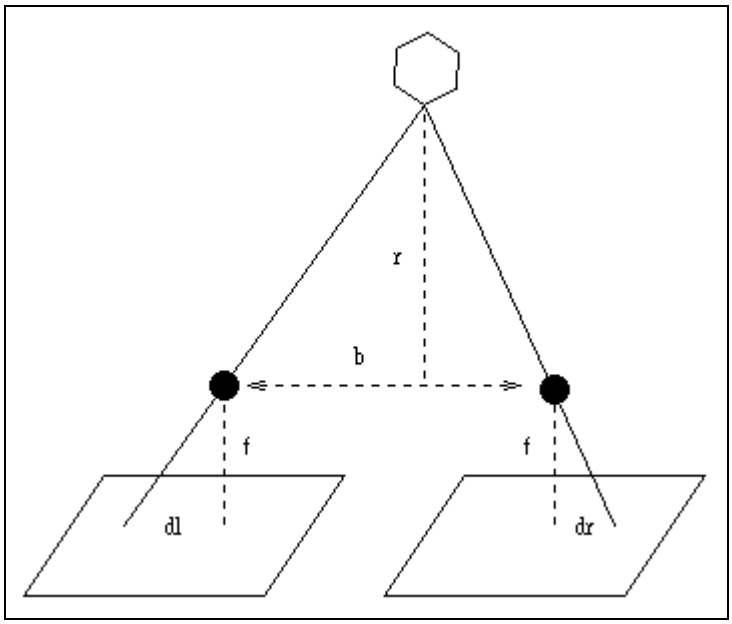


Figure 3-1. Definition of disparity: offset of the image location of an object.

Normally, we set up the stereo cameras so that their image planes are embedded within the same plane. Under this condition, the difference between **dl** and **dr** is called the *disparity*, and is directly related to the distance **r** of the object normal to the image plane. The relationship is:

(1) **r = bf / d** , where **d = dl - dr** .

Using Equation 1, we can plot range as a function of disparity for the STH-V1 stereo head. At their smallest baseline, the cameras are about 8 cm apart. The pixels are 14 um wide, and the standard lenses have a focal length of 6.3 mm. For this example, we get the plot in Figure 3-2. The minimum range in this plot is 1/2 meter; at this point, the disparity is over 70 pixels; the maximum range is about 35 meters. Because of the inverse relationship, most of the change in disparity takes place in the first several meters.

The range calculation of Equation (1) assumes that the cameras are perfectly aligned, with parallel image planes. In practice this is often not the case, and the disparity returned by the Stereo Engine will be offset from the ideal disparity by some amount X_0. The offset is explained in the section below on the horopter, and in the section on calibration.

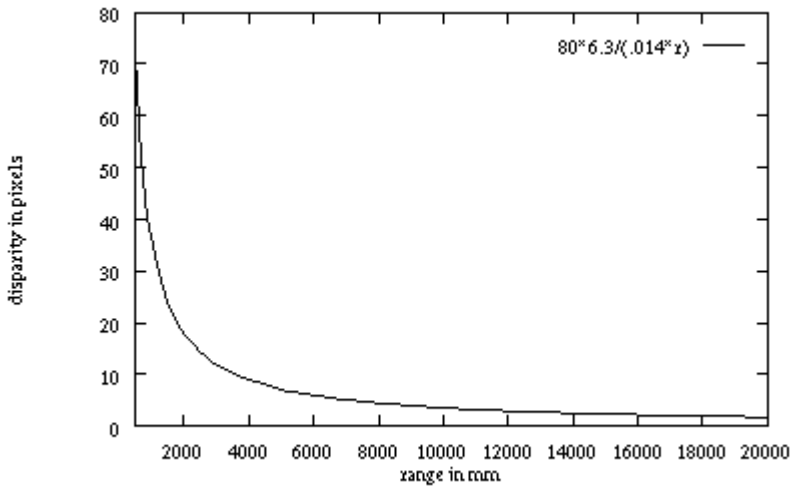


Figure 3-2. Inverse relationship between disparity and range. This plot is for a focal length of 6.3 mm, a baseline of 80 mm, and a pixel width of 14 mm.

3.2 Horopter

Stereo algorithms typically search only a window of disparities, e.g., 16 or 32 disparities. In this case, the range of objects that they can successfully determine is restricted to some interval. The *horopter* is the 3D volume that is covered by the search range of the stereo algorithm. The horopter depends on the camera parameters and stereo baseline, the disparity search range, and the X offset. Figure 3-3 shows a typical horopter. The stereo algorithm searches a 16-pixel range of disparities to find a match. An object

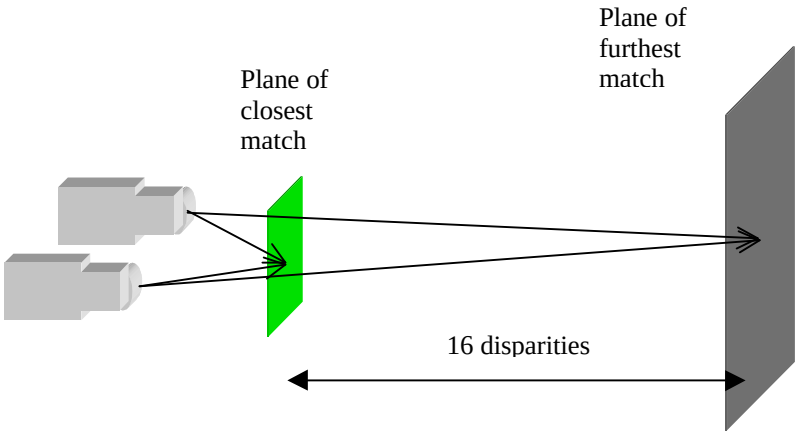


Figure 3-3 Horopter planes for a 16-pixel disparity search.

that has a valid match must lie in the region between the two planes shown in the figure. The nearer plane has the highest disparity (15), and the farthest plane has the lowest disparity (0).

The placement of the horopter can be varied by changing the X offset between the two images, which essentially changes the search window for a stereo match. Figure 3-5 shows the raw disparities for a typical stereo head. The cameras are slightly verged, so a zero disparity plane (where an object appears at the same place in both images) occurs at some finite distance in front of the cameras. If the stereo algorithm is searching 5 disparities, then without any X offset, it will search as shown in the top red arrow, that is, from disparity 0 to disparity 4. By offsetting one image in the X direction by *n* pixels, the horopter can be changed to go from *–n* to *5-n* raw disparities. This search range is indicated by the lower red arrow.

Generally, it is a good idea to set the X offset to compensate for camera vergence or divergence, that is, to set it so that the furthest horopter plane is at infinity. The reason that this is a good idea is because it's usually possible to control how close objects get to the camera, but not how far away. The offset that puts the far horopter plane at infinity is called X_0. With this offset, a disparity of 0 indicates an infinitely far object.

The horopter can be determined from Equation (1). For example, if the disparity search window is 0-31, the horopter (using the graph above) will be from approximately 1 meter to infinity. The search window can be moved to an offset by shifting the stereo images along the baseline. The same 32 pixel window could be moved to cover 10-41 pixel disparities, with a corresponding horopter of 0.8 meters to 2.2 meters.

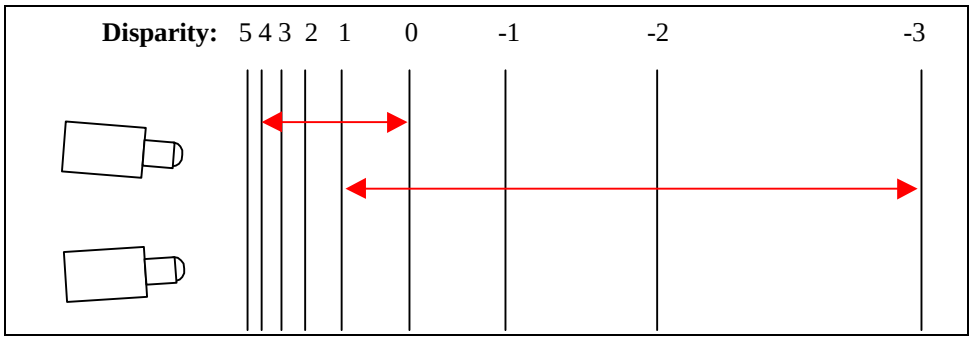


Figure 3-5. Planes of constant disparity for verged stereo cameras. A search range of 5 pixels can cover different horopters, depending on how the search is offset between the cameras.

The location and size of the horopter depends on the application. If an object falls outside the horopter, then its true disparity will not be found, and instead it will get some random distribution of disparities. Figure 3-4 shows what happens when the object's range falls outside the horopter. In the left image, the disparity search window is correctly positioned so that objects from 1 meter to infinity are in view. In the right image, the window has been moved back so that objects have higher disparities. However, close objects are now outside of the horopter, and their disparity image has been "broken up" into a random pattern. This is typical of the disparity images produced by objects outside the horopter.

For a given application, the horopter must be large enough to encompass the ranges of objects in the application. In most cases, this will mean positioning the upper end of the horopter at infinity, and making the search window large enough to see the closest objects.

The horopter is influenced not only by the search window and offset, but also by the camera parameters and the baseline. The horopter can be made larger by some combination of the following:

- Decreasing the baseline.
- Decreasing the focal length (wider angle lenses).
- Increasing pixel width.
- Increasing the disparity search window size.

As the cameras are moved together, their viewpoints come closer, and image differences like disparity are lessened. Decreasing the focal length changes the image geometry so that perceived sizes are smaller, and has a similar effect. It also makes the field of view larger, which can be beneficial in many applications. However, very small focal length lenses often have significant distortion that must be corrected (see the section on calibration). Another way to change the image geometry is to make the pixels wider. This can be done by scaling the image, e.g., from 320x240 to 160x120, which doubles the pixel size. Note that it is only necessary to change the pixel width. Most framegrabbers have hardware scaling

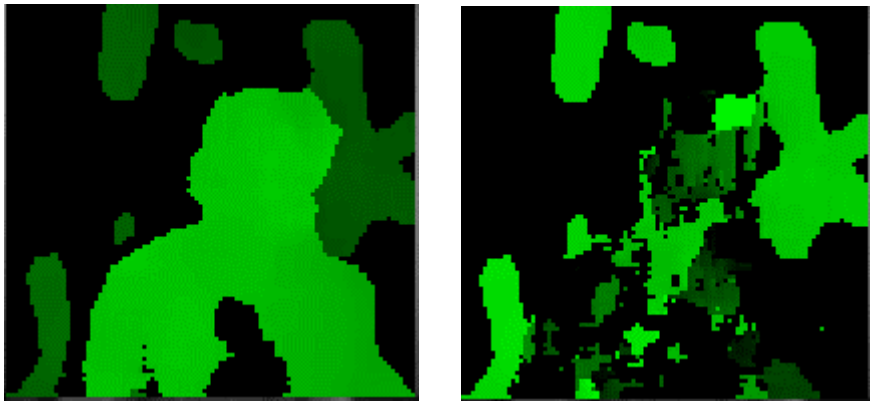


Figure 3-4. Disparity image for all regions withing the horopter (left) and some regions outside the horopter (right).

to arbitrary resolutions.

These first three options change the camera geometry, and thus have a corresponding effect on the range resolution, which decreases (see below). The only way to increase the horopter size and maintain range resolution is to increase the disparity search window size, which leads to more computation. Multiresolution methods, which use several sizes of an image, each with its own horopter, are one way to minimize computation (see, for example, the paper by Iocchi and Konolige at *www.ai.sri.com/~konolige/svs*).

3.3 Range Resolution

Often it's important to know the minimal change in range that stereo can differentiate, that is, the *range resolution* of the method. Give the discussion of stereo geometry above, it's easy to see that that range resolution is a function of the range itself. At closer ranges, the resolution is much better than farther ranges.

Range resolution is governed by the following equation.

(2) $\Delta \mathbf{r} = (\mathbf{r}^2/\mathbf{bf}) \Delta \mathbf{d}$

The range resolution, $\Delta\mathbf{r}$, is the smallest change in range that is discernable by the stereo geometry, given a change in disparity of $\Delta\mathbf{d}$. The range resolution goes up (gets worse) as the square of the range. The baseline and focal length both have an inverse influence on the resolution, so that larger baselines and focal lengths (telephoto) make the range resolution better. Finally, the pixel size has a direct influence, so that smaller pixel sizes give better resolution. Typically, stereo algorithms can report disparities with subpixel precision, which also increases range resolution.

The figure below plots range resolution as a function of range for the STH-MD1 (MEGA-D) stereo head, which has a baseline of 9 cm. The Stereo Engine interpolates disparities to 1/16 pixel, so $\Delta\mathbf{d}$ is 1/16 * 7.5 um = 0.08533 um. The range resolution is shown for a sampling of different lens focal lengths. At any object distance, the range resolution is a linear function of the lens focal length.

Equation 2 shows the range resolution of a perfect stereo system. In practice, video noise, matching errors, and the spreading effect of the correlation window all contribute to degrading this resolution.

Range resolution is not the same as range accuracy, which is a measure of how well the range computed by stereo compares with the actual range. Range accuracy is sensitive to errors in camera calibration, including lens distortion and camera alignment errors.

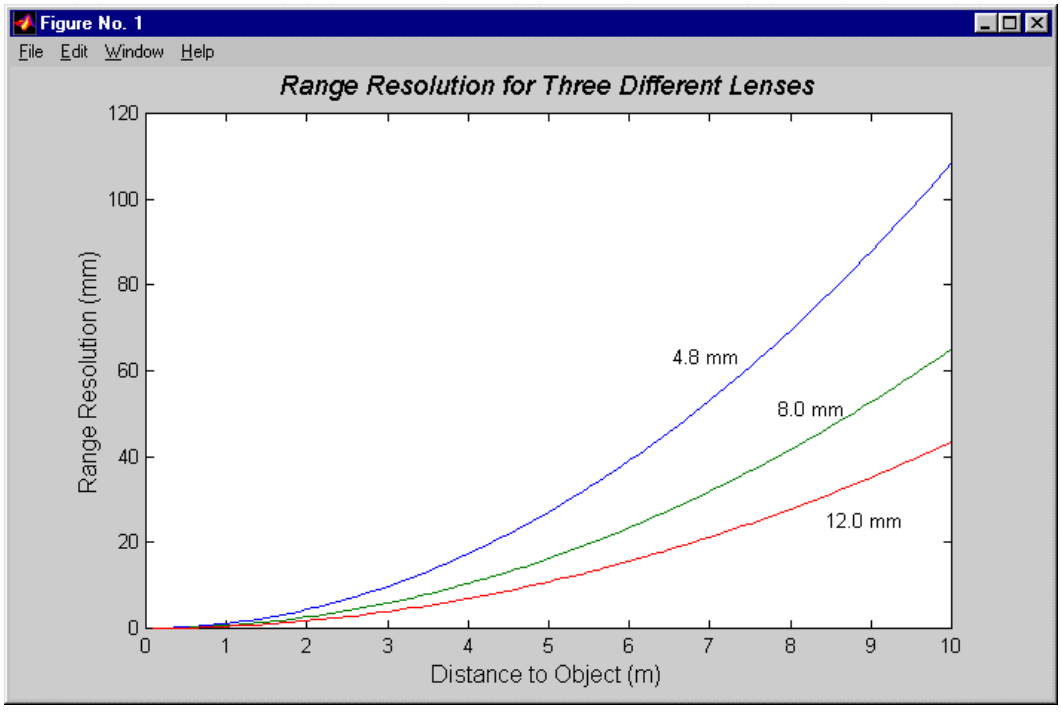


Figure 3-6. Range resolution as a function of range. This plot assumes a baseline of 90 mm, and a pixel size of 7.5 um, with subpixel resolution of 1/16 pixel.

3.4 Area Correlation Window

Stereo analysis is the process of measuring range to an object based on a comparison of the object projection on two or more images. The fundamental problem in stereo analysis is finding corresponding elements between the images. Once the match is made, the range to the object can be computed using the image geometry.

Area correlation compares small patches, or *windows*, among images using correlation. The window size is a compromise, since small windows are more likely to be similar in images with different viewpoints, but larger windows increase the signal-to-noise ratio. Figure 3-7 shows a sequence of disparity images using window sizes from 7x7 to 13x13. The texture filter was turned off to see the effects on less-textured areas, but the left/right check was left turned on.

There are several interesting trends that appear in this side-by-side comparison. First, the effect of better signal-to-noise ratios, especially for less-textured areas, is clearly seen as noise disparities are eliminated in the larger window sizes. But there is a tradeoff in disparity image spatial resolution. Large windows tend to "smear" foreground objects, so that the image of a close object appears larger in the disparity image than in the original input image. The size of the subject's head grows appreciably at the end of the sequence. Also, in the 7x7 the nose can be seen protruding slightly; at 13x13, it has been smeared out to cover most of the face.

One of the hardest problems with any stereo algorithm is to match very small objects in the image. If an object does not subsume enough pixels to cover an appreciable portion of the area correlation window, it will be invisible to stereo processing. If you want to match small objects , you have to use imagers with good enough spatial resolution to put lots of pixels on the object.

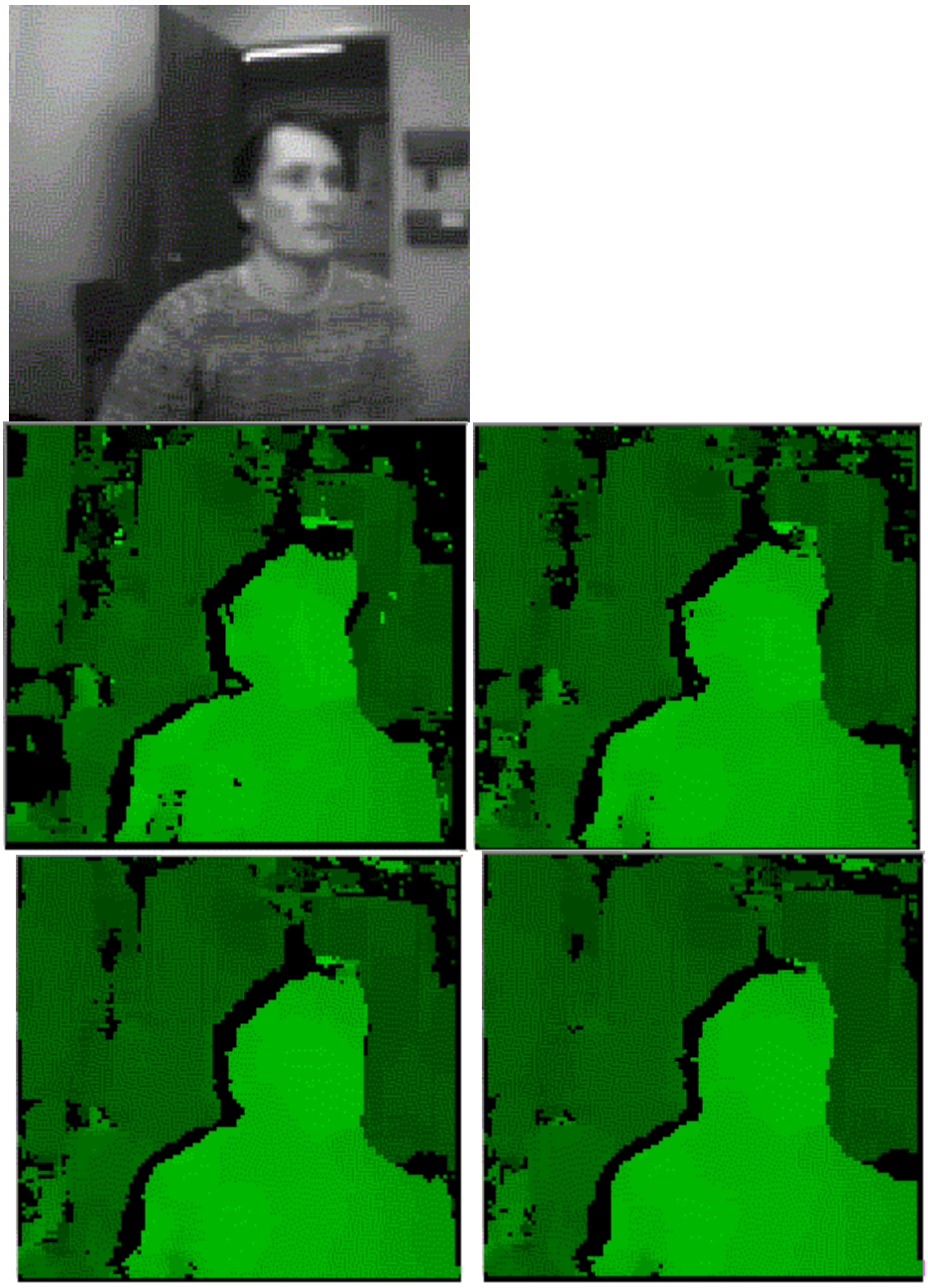


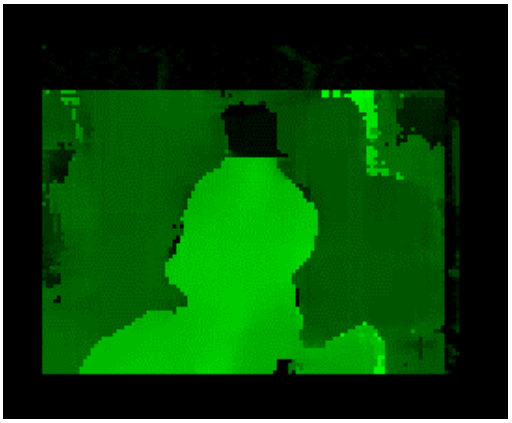
Figure 3-7 Effects of the area correlation window size. At top is the original left intensity image. The greenscale images show windows of 7x7, 9x9, 11x11, and 13x13 windows (clockwise from upper left).

3.5 Filtering

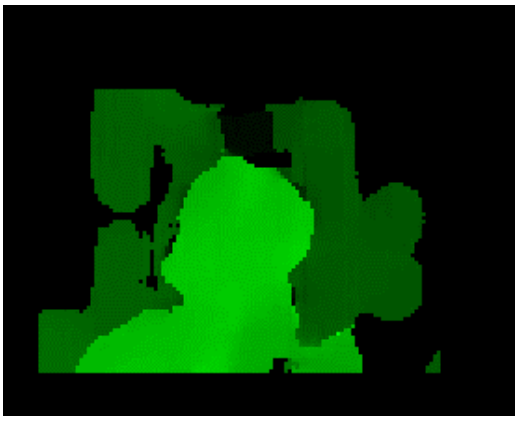
Like most vision algorithms, the results of stereo processing can contain errors. In the case of stereo, these errors result from noisy video signals, and from the difficulty of matching untextured or regularly textured image areas. Figure 3-8 shows a typical disparity image produced by the SRI algorithm. Higher disparities (closer objects) are indicated by brighter green (or white, if this paper is printed without color). There are 64 possible levels of disparity; in the figure, the closest disparities are around 40, while the furthest are about 5. Note the significant errors in the upper left and right portion of the image, where uniform areas make it hard to estimate the disparity.

In Figure 3-8(c), the interest operator is applied as a postfilter. Areas with insufficient texture are rejected as low confidence: they appear black in the picture. Although the interest operator requires a threshold, it's straightforward to set it based on noise present in the video input. Showing a blank gray area to the imagers produces an interest level related only to the video noise; the threshold is set slightly above that. Or, more simply, you can use the temporal variance of poorly textured matches to adjust the texture threshold. Observing the disparity image during realtime display, there will usually be areas that flicker rapidly. Adjust the threshold upward until these regions disappear. If there are no such regions, adjust the threshold downward until just before they appear.

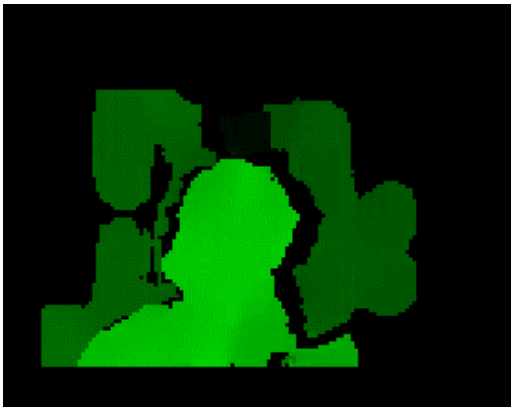
(a) Input grayscale image, one of a stereo pair



(b) Disparity image from area correlation



(c) Texture filter applied



(d) Left/right and texture filter applied

Figure 3-8 Post-filters applied to a disparity image. (c) is a texture filter that eliminates textureless areas. (d) is a consistency check between left and right stereo matches.

There are still errors in portions of the image with disparity discontinuities, such as the side of the subject's head. These errors are caused by overlapping the correlation window on areas with very different disparities. Application of a left/right check can eliminate these errors, as in Figure 3-8(d). The

left/right check can be implemented efficiently by storing enough information when doing the original disparity correlation.

In practice, the combination of an interest operator and left/right check has proven to be the most effective at eliminating bad matches.

3.6 Performance

Using standard PC hardware, running either MS Windows 95/98/2000/NT or Linux, the SVS can compute stereo range in real time. Table 3-1 gives some typical timings for a 500 MHz Pentium III processor. Because the Stereo Engine has a very small memory footprint, the timings scale almost linearly with increasing processor speed. These timings include the complete stereo algorithm detailed above: dewarping of input images, disparity computation and interpolation, and post-filtering using a texture filter and left/right filter.

Frame size	Number of Disparities	Frame Rate
160x120	16	180 Hz
160x120	32	100 Hz
320x240	16	45 Hz
320x240	32	24 Hz
640x480	32	6 Hz

Table 3-1 Processing rates on a Pentium III 500 MHz machine.

4 Calibration

Most stereo camera setups differ from an ideal setup in which the cameras are perfect pinhole imagers and are aligned precisely parallel. The divergence from ideal causes problems in the quality of the stereo match since epipolar lines are not horizontal. In addition, if the camera calibration is unknown, one does not know how to interpret the stereo disparities in terms of range to an object. Camera calibration addresses these issues by creating a mathematical model of the camera.

SVS incorporates a simple automatic procedure for calibration, using a planar object that can be printed on a standard printer. The calibration is preformed by fitting a model to a number of images taken of a planar calibration object. The user presents the object to the stereo rig in five different (arbitrary) poses. The calibration procedure finds model features in the images, and then calculates a best-fit calibration for the rig. The procedure works for many different combinations of imagers, baselines, and lenses, including wide-angle lenses with severe distortion.

When is it necessary to perform calibration? In general, whenever an action changes the camera intrinsics (lens focal length and center axis) or extrinsics (the cameras move with respect to each other). Here are some actions that would necessitate re-calibration:

- Changing lenses
- Screwing the lenses in or out of their mount
- Zooming, if the lenses are zoom lenses
- Changing the baseline of the cameras
- Any movement or rotation of one camera independent of the other, e.g., severe vibration or shock can change the cameras' relative position

A rigid mount that keeps the cameras stable with respect to each other is a necessity for a stereo rig. For example, the MEGA-D uses an extruded aluminum frame to stabilize the cameras. There are some actions that do *not* require re-calibration:

- Changing the lens focus with a focusing ring on the lens
- Changing the lens aperture

The next section reviews the calibration procedure, detailing the steps required to generate a calibration file. Section **Error! Reference source not found.** is a more technical description of calibration and stereo geometry.

4.1 Calibration Procedure

An automatic calibration procedure using five image pairs of a planar calibration target is included as part of the `smallv` program. Given the image pairs of the calibration object, the system automatically locates corner features in the target, fits a model of the target to the images, and finally produces an estimate for the left and right camera intrinsics, the stereo head extrinsics, and the rectification matrices $\mathbf{P}_0$, $\mathbf{P}_1$, $\mathbf{H}_0$, and $\mathbf{H}_1$. These values are then used by `smallv`, and can also be saved as a parameter file for later re-use. More information about the calibration procedure can be found in the Calibration Addendum to the User's Manual.

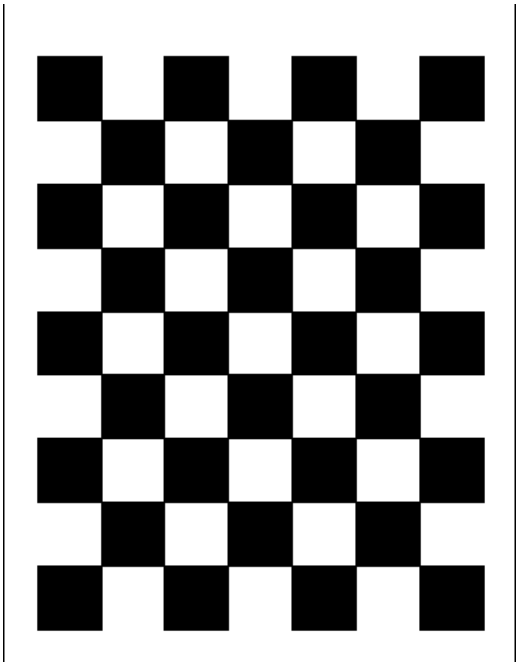


Figure 4-1 Checkerboard calibration object.

4.1.1 Calibration procedure steps

1. Create the calibration object. Print out a copy of the file *`data/check.pdf`* (Figure 4-1) and paste it to a surface that is as flat as possible. We use a wooden cutting board as a backing surface at SRI.
2. Start the `smallv` application and start capturing video. It is recommended that you set the video resolution to at least 320x120 in order to get enough detail of the calibration object. A calibration computed when capturing video at a higher resolution can be used for future video captured at any resolution with the same cameras.
3. Bring up the calibration window by pressing the `Calibrate`... menu button. Fig. 4-4 shows the calibration dialog window (the figure shows the dialog after an image has been captured and processed).
4. Determine the appropriate characteristics of the camera imagers and enter them into the four boxes in the middle of the dialog. If you have one of the Videre Design stereo heads, check the appropriate box and the parameters are loaded automatically.
5. Acquire five stereo pairs of the calibration object at different rotations and translations. Try to avoid views that differ by a simple translation, as they are less informative than views with variation in rotation. As shown in Fig 4-4, there is a tab control that shows only one pair at a time; choose a tab to select another pair. To capture the current video feed into a stereo pair box, simply press the capture button. You can also save and load images to and from disk using the `load` and `save` buttons.
6. Detect the corner features in all views by pressing the `features` button in the lower bar of buttons. This cycles through all the images, displaying the corners in green as they are detected. If the feature finder fails on an image, please re-capture the image and redetect the features. When redetecting features for a single stereo pair, use the `features` button in the stereo pair box.
7. Compute the calibration parameters (intrinsics, extrinsics) and rectification matrices by pressing the `calibrate` button in the lower button bar. This operates in three phases:
   a) Calibrates individual views using a planar model of the calibration object. The projection of these model features is shown in red.
   b) Calibrates all the views jointly using nonlinear optimization over all the intrinsic and extrinsic parameters. This phase usually takes a few minutes, and when finished, the projected model features are shown in yellow.

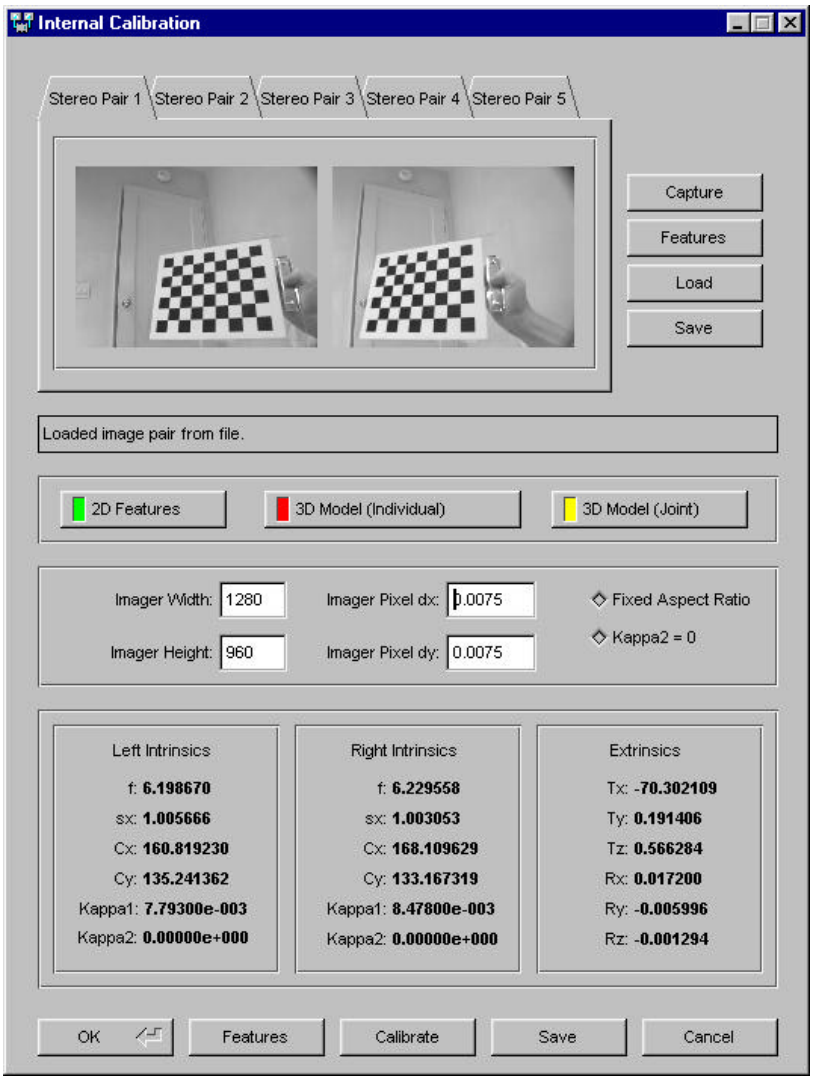


Figure 4-4 Smallv calibration dialog window.

c) Computes the rectification matrices from the joint calibration in (b).

7. When the calibration is finished, the parameter listing at the bottom is updated, and you have a couple additional options in the lower button bar: `save` writes the parameter file to disk and `OK` exits the calibration dialog and transfers the new parameters to the main `smallv` window.

4.1.2 Imager Characteristics

The calibration process must be related to the geometry of the camera imagers. There are three important parameters:

1. Pixel size in mm (width and height)
2. The width of the imager in pixels
3. The width of the image output by the framegrabber

The pixel size can be found from the specifications of the imager. If you cannot find these, just use defaults of 0.010 mm. The calibration will determine the pixel aspect ratio (width / height). The calibrated lens focal length will not be correct because the pixel scale will be off, but that will not change the validity of the calibration.

The imager width is the number of sels (sensor elements) in each line of the imager. Again, this can be found in the imager specifications. If these are not available, just use the image width as it comes from the framegrabber, e.g., for NTSC video it is 640.

The framegrabber width is the width of the image output by the framegrabber, in pixels. Typically it will be an analog NTSC signal, which is 640 pixels. For digital imagers, such as the MEGA-D, the image size in sels and the framegrabber width are the same.

5 API Reference

To add stereo processing to your own programs, you call functions in the Stereo Engine library. These functions are available in `svs.dll` (Windows 95/98/2000/NT) or `libsvs.so` (Unix systems). The header file is `src/svs.h`. The current version of the library is 2.1. A simple example of the use of these functions is in the sample program `samples/stframe.c`.

5.1 Stereo Parameter Structure

The parameter structure `svsSP` holds information about the whole stereo process, from camera characteristics, the images input from the cameras, and the parameters involved in controlling the stereo process. Almost every function in the API refers to a parameter structure.

By default, there is a global parameter structure `svsParams` in `svs.dll` that is used whenever the parameter structure is not specified in a function.

```
typedef struct {
  int id;                    /* can be set by user */

  /* Stereo algorithm parameters */
  int convx, convy;          /* Edge convolution kernel size, pixels */
  int corrxsize, corrysize;  /* Correlation window size, pixels */
  int thresh;                /* Confidence threshold, 0-20 */
  int lr;                    /* Left/right check, 1=on, 0=off */
  int ndisp;                 /* Number of pixel disparities to search */
  int dpp;                    /* Disparities per pixel */
  int offx, offy;             /* Horopter offset (offy not used) */

  /* framegrabber/image parameters absolute parameters */
  int max_linelen;           /* for NTSC fields, 640 */
  int max_lines;             /* for NTSC fields, 240 */
  int max_decimation;        /* 1, 2 or 4 */
  int max_binning;           /* 1 or 2 */
  double gamma;              /* gamma correction needed for display */
  int color;                 /* 0 for monochrome images, 1 for color */

  /* Current full-frame image size */
  int linelen;               /* Image line length, in pixels */
  int lines;                  /* Number of image lines */
  int decimation;            /* current decimation */
  int binning;               /* current binning */
  int subwindow;             /* 1 = subwindow capability, 0 = none */

  /* Image subwindow */
  int ix;                    /* Subimage start column */
  int iy;                    /* Subimage start row */
  int width;                 /* Subimage width, in pixels */
  int height;                /* Subimage height, in pixels */
  int vergence;              /* Subimage vergence between images */


  /* digitization parameters */
  int autogain;              /* 1 if auto gain available */
  int manualgain;            /* 1 if manual gain available */
  int autowhite;             /* 1 if auto white balance available */
  int manualwhite;           /* 1 if manual white balance available */
  int gain;                  /* Image gain, from 0 to 100; -1 is auto */
  int exposure;              /* Image exposure from 0 to 100, -1 is auto
*/
  int contrast;        /* Image contrast, from 0 to 100, -1 is auto */
  int brightness;      /* Image brightness, from 0 to 100, -1 is auto */
  int saturation;            /* Image color saturation, from 0 to 100 */
  int red;                       /* red gain offset, from –40 to 40 */
  int blue;                  /* blue gain offset, from –40 to 40 */

  /* Internal camera optical parameters */
  svsIP  left;               /* Left camera parameters */
  svsIP  right;              /* Right camera parameters */

  /* Transformation between left and right cameras */
  /* Coordinate system is attached to the center of projection of the
     left camera, with the X and Y axis aligned with the u and v axis,
```

```
        and Z along the line of sight */
      double Tx;                  /* [mm] */
      double Ty;                  /* [mm] */
      double Tz;                  /* [mm] */
      double Rx;                  /* [rad] Yaw */
      double Ry;                  /* [rad] Pitch */
      double Rz;                  /* [rad] Roll */

      unsigned long reserved;
    } svsSP;

    typedef struct {
      /* Intrinsic parameters */
      int       pwidth;    /* [pix]    Width of frame grabber's image */
      int       pheight;   /* [pix]    Height of frame grabber's image */
      double    dpx;       /* [mm/pix] X dimension of pixel in frame grabber */
      double    dpy;       /* [mm/pix] Y dimension of pixel in frame grabber */
      double    sx;        /* []       Scale factor to compensate for any error
in dpx   */
      double    Cx;        /* [pix]    Z axis intercept of image plane */
      double    Cy;        /* [pix]    Z axis intercept of image plane */
      double    f;         /* [mm]     Focal length */
      double    kappa1;    /* [1/mm^2] First coefficient of radial distortion
*/
      double    kappa2;    /* [1/mm^4] Second coefficient of radial distortion
*/
      float     proj[3][4]; /* projection matrix of rectified image */
      float       rect[3][3]; /* rectification transform after correction for
lens distortion */
    } svsIP;
```

5.2 Acquiring images from the video capture board

The SVS library includes functions for interfacing to various video capture boards. These functions are written for each board, to present a standard interface. Several global variables are used to determine the parameters of video capture.

5.2.1 Video Capture Interface Libraries

Video is captured through a framegrabber, either analog or digital. The framegrabber must be installed according to instructions for the particular board and OS. For MS Windows, these instructions come with the framegrabber. In Linux, there are installations for Bt8xx boards, Matrox Meteor boards, and 1394 digital boards. Links to relevant websites for installation can be found on the Videre Design website (www.videredesign.com).

Once a framegrabber is installed, SVS can access it via capture interface libraries. To use a particular interface, it must be copied to `bin/svsgrab.so` (Linux) or `bin/svsgrab.dll` and `bin/svsgrab.lib` (MS Windows). The following table lists the interface libraries for different framegrabbers.

MSW 95/98/2000/NT	Imagenation PXC200	`mswpxc.dll, lib`
	Matrox Meteor Standard Matrox Meteor PPB, RGB	`svsmet.dll, lib`
	Matrox Meteor II	`svsmet2.dll, lib`
	VPP PCMCIA	`svsvpp.dll, lib`
	VFW framegrabber	`svsvfw.dll, lib`
MSW 98SE/2000	1394 OHCI digital	`svspix.dll, lib`
Linux	Imagenation PXC200 Matrox Meteor Standard Matrox Meteor PPB, RGB	`metcap.so`
Linux 2.2.14 and above	1394 OHCI digital	`pixcap.so`

Table 5-1 Framegrabber interface libraries

5.2.2 Writing a Framegrabber Interface Library

Examples of these libraries are found in `src/mswXXX.c` and `src/lnxXXX.c`. Use these examples as a template for writing a framegrabber interface library for a new framegrabber.

5.2.3 Opening the Capture Device

Before video capture can begin, the video devices must be opened. The video capture parameters are set via global parameters, and then `svsVideoOpen` is called.

`int svsVideoOpen(svsSP *sp)`

Opens the video capture device or devices (depending on `svsDualCaptureFormat`). Updates the camera parameters of `sp`, based on the values assigned by the interface, e.g., sets the maximum decimation and binning, whether the cameras have auto gain or auto white balance, etc.

It is not necessary to have valid values for window size or digitization parameters when the device is opened. These values must be set before starting the video stream (`svsVideoStartContinuous`).

Returns 1 if successful, and 0 if not.

`void svsVideoClose(void)`

Closes the connection to the video capture device(s). Usually not needed, since the connection will be closed when a program exits. However, it's nice to do this, and some drivers under MS Windows are less forgiving.

5.2.4 Capture Device Format Parameters

These global parameters dictate the basic format for analog framegrabber devices. They should be set before calling the `svsVideoOpen` function. The default formats are for the STH-V3 system.

`int svsDualCaptureFormat`

A global integer that determines the type of video capture system. Values are:

- `CAP_INTERLACE` Single video stream with alternate lines containing left and right images.
- `CAP_DUAL` Dual video stream requiring two capture boards.
- `CAP_RGCOMPONENT` Dual video stream using a single RGB capture board. Left image is on the R component, right on the G component.

This parameter is not relevant for the MEGA-D digital system.

`int svsHasFields`

A global integer that determines whether the capture board driver can save single fields to memory. Values are 0 (only interlaced frames) or 1 (can save fields). Not relevant for the MEGA-D digital system.

`int svsVideoFormat`

A global integer that determines the pixel format of the video stream captured to memory. Only used in Video for Windows drivers. The two supported values are:

- `BI_YVU9` 9 bits/pixel, with reduced color information. This is the typical format for interlaced and dual capture.
- `BI_RGB` 24 bits/pixel RGB component. This value is only used for RGB component capture cards.

`int svsVideoBPP`

A global integer that determines the number of bits per pixel. Only used with Video for Windows drivers. Should be set to 9 for `BI_YVU9`, and 24 for `BI_RGB`.

5.2.5 Frame Size Parameters

Most framegrabbers support some way to specify the image size, that is, the height and width of the returned image. The cameras themselves output an analog signal that has a native resolution, for example, NTSC cameras have a native resolution of 640 x 480. For most analog framegrabbers, the image size implicitly determines the subsampling of the image, e.g., specifying a 320x240 image size means the framegrabber subsamples a 640x480 (NTSC) camera image by 2.

Some analog framegrabbers also support subwindowing, where only a portion of the image is actually digitized and returned. There is no current support in the SVS interface libraries for analog framegrabber subwindowing, although it would not be difficult to add.

Some cameras, especially digital cameras, support direct control over the number of pixels returned in each image. For these cameras, explict specification of sampling modes and subwindowing is possible, and the 1394 OHCI interface library for the MEGA-D supports these modes.

All of the frame size parameters should be set after opening the stereo head with `svsVideoOpen`, and before returning a video stream with `svsVideoStartContinuous`. Setting the size parameters during streaming video will have no effect, until the streaming is stopped and `svsVideoStartContinuous` is called again. The exceptions are the offset and vergence of a subwindow, and color information, which can be changed during video streaming.

```
int max_linelen
int max_lines
int subwindow
```

These parameters are filled in by the framegrabber interface, and should not be changed by the user. They specify the maximum available height and width from the stereo cameras, in pixels. The `subwindow` flag is 1 if subwindowing of the image is available in hardware, and 0 if not.

```
int linelen, lines
int width, height
int ix, iy
int vergence
```

`linelen` and `lines` specify the current width and height of the full-frame image available from the cameras. This will not necessarily be the same as `max_linelen` and `max_lines`, since the available images may be downsampled. For example, an NTSC analog framegrabber allows for an arbitrary downsampling from 640x480.

`height` and `width` are the size of the image currently returned from the cameras. Normally, these will be the same as `linelen` and `lines`, unless the cameras support subwindow mode; NTSC analog framegrabbers do not, but the MEGA-D digital stereo head does.

To set up a subwindow with the MEGA-D digital head, set the window size with svsVideoSetSize to be less than the full-frame size for the sampling mode. For example, if there is no subsampling, then setting a window size of 640x480, for example, will return a 640x480 subwindow of the full 1288x1032 image.

`ix` an `iy` are the subwindow offsets from the upper left corner of the image, in pixels. If subwindowing is not used, they will be zero. The values for these parameters are limited by the values of the subwindow width and height relative to the full-frame size.

Electronic vergence allows the stereo head to keep near objects centered in both images. Vergence can be performed in subwindow mode by setting a differential between the X offsets of the images. `vergence` is specified in pixels, and is accomplished in the MEGA-D by offsetting the right image subwindow further to the right.

```
int svsVideoSetOffsets(svsSP *sp)
```

Checks the subwindow offsets and vergence in `sp`, and if any of them have changed since the last call to this function or to `svsVideoStartContinuous`, it updates them on the stereo hardware. Returns 0 if the call succeeds, -1 if not.

```
int max_decimation
int max_binning
```

These parameters indicate the maximum decimation and binning values supported by the cameras. They are set by the interface during `svsVideoOpen`, and should not be changed by the user.

If max_decimation is 1, then the stereo cameras don't support explicit decimation. A value of 2 means the cameras can decimate by 2 horizontally and vertically, and a value of 4 means they can decimate by 2 or 4. For example, the MEGA-D has a decimation value of 4, which means that it supports image sizes of 1288 x 960, 640 x 480, and 320 x 240. Note that most analog framegrabbers support implicit subsampling, by setting the frame size. These functions are used by digital cameras like the MEGA-D that support subsampling at the imager.
If max_binning is 1, then the stereo cameras don't support explicit binning. Binning is like decimation, but it averages a group of pixels. A value of 2 means the cameras can decimate by 2, and a value of 4 means they can decimate by 2 or 4. These functions are used by digital camera that support subsampling at the imager. For the MEGA-D, the interface library supports software binning by a factor of 2.

```
int decimation
int binning
```

These are the current values for decimation and binning. They should be set before calling svsVideoStartContinuous.

```
int color
```

This parameter controls whether or not color information is made available. If it is 0, no color information is returned to the user program from svsVideoGetImage. If it is 1, and the stereo head has color capability, a color image is returned.

5.2.6 Digitization Parameters

Some framegrabbers and cameras support control over the appearance of an image, the relationship between light reaching the sensor and the pixel value produced. Most cameras have automatic exposure and gain control; the framegrabber can set brightness and contrast as it digitizes the analog signal. The MEGA-D has manual exposure and gain.

In general, the digitization parameters can be changed at any point, even when the video is streaming to the user program. The special function svsVideoSetDigParams will immediately apply all digitization settings that have changed since the start of video streaming, or the last call to svsVideoSetDigParams.

```
int svsVideoSetDigParams(svsSP *sp)
```

Checks the digitization parameters in sp, and if any of them have changed since the last call to this function or to svsVideoStartContinuous, it updates them on the stereo hardware. Returns 0 if the call succeeds, -1 if not.

```
int autogain
int manualgain
int autowhite
int manualwhite
```

These parameters give information about the digitization modes available from the cameras. They are set by the interface software when the cameras are opened with svsVideoOpen, and should not be changed by the user.
If autogain is 1, the cameras can set gain and exposure automatically to compensate for changing lighting conditions. If manualgain is 1, the user can set exposure and gain under program control. These are not necessarily exclusive: some cameras support both auto

and manual gain. The shift from auto to manual is controlled by the value set on `gain` or `exposure`.
Similarly, some color cameras can automatically balance the image colors (`autowhite`), and some cameras allow manual setting of color balance (`manualwhite`). Values of 1 indicate the capability is available on the cameras.

`int color`

If set to 1, then color information is returned by calls to `svsVideoGetImage`, if the stereo head supports it.
If set to 0, then no color information is returned.

`int gain`
`int exposure`

If either gain or exposure is set to –1, then auto gain is used. If auto gain is not available, then setting either to –1 will cause an error in `svsVideoCheckParameters` or `svsVideoSetDigParams`.
For manual control, exposure and gain are set on a scale of 0 to 100. For best image quality, it is preferable to set the exposure high and the gain low.

`int contrast`
`int brightness`

For cameras with auto gain, the auto gain performance can be tuned using these two parameters. They range from 0 to 100, with nominal values of 50. Higher `contrast` values increase the contrast of the auto gain facility, while lower numbers decrease it. Similarly, the `brightness` value will cause the auto gain to settle on a brighter or darker image.
These parameters only have effect if the camera supports auto gain, and the `gain` or `exposure` parameter is set to –1, indicating auto gain mode is in effect.

`int red`
`int blue`

These parameters are used to set the color balance of the cameras in manual white balance mode. They range from –40 to 40. Negative values lower the amount of the corresponding color in an image.

5.2.7 Checking Image Parameters

Before starting a continuous video stream, the SVS system requires that the image parameters be consistent. In general, this means that the image size, decimation/binning, subwindowing, and vergence are consistent and implemented by the stereo head. The function `svsVideoCheckParameters` looks at an `svsSP` structure, and returns an error code if the parameters are not consistent. It is recommended that `svsVideoCheckParameters` be called before each call to `svsVideoStartContinuous`.

`int svsVideoCheckParameters(svsSP *sp)`

Returns an error code if the parameter structure `sp` is not consistent or implementable; otherwise returns 0.
Error codes (defined in `svs.h`):

`VIDEO_ERROR_DEC_MODE`	Decimation mode is not available
`VIDEO_ERROR_BIN_MODE`	Binning mode is not available
`VIDEO_ERROR_FRAME_SIZE`	Frame width or height is not consistent
`VIDEO_ERROR_SUBWINDOW`	Subwindow size is not consistent

5.3 Continuous Capture

In continuous capture mode, the capture device sends images to main memory at video rates. A double-buffering scheme lets user programs run asynchronously. When the user program wants a stereo pair, it requests it with the sequence `svsVideoWaitImage - svsVideoGetImage`. This will give the program a pointer to the most recent acquired stereo pair. The buffer is managed by SVS, so the application should not free it up after processing.

```
int svsVideoStartContinuous(svsSP *sp)
```

Starts continuous video capture. The video device(s) must already have been opened with `svsVideoOpen`. Images are grabbed continuously into a dual buffer at frame rates, and can be retrieved asynchronously with `svsWaitImage`/`svsGetImage`.

When this function is called, all the frame size and digitization parameters in `sp` are checked, and the cameras hardware is set up according to these parameters. The major reason for an unsuccessful call is that the frame size parameters are not legal. Use the `svsVideoCheckParams` function to check the validity of parameters before calling `svsVideoStartContinuous`.

Returns 0 if successful, and a negative error number if the video stream could not be started. The error codes are the same as for `svsVideoCheckParameters`, defined above.

```
void svsVideoHaltContinuous(void)
```

Halts continuous video capture.

```
int svsVideoWaitImage(int ms)
```

Waits for a frame in continuous capture mode. The ms argument is an integer specifying the timeout period in milliseconds. Returns 1 if an image is ready, and 0 if it timed out.

Normally, because of double-buffering of the continuous image stream, this function will return 1 immediately. Only in the case where the user program requests images faster than the video rate will it actually wait before returning.

After a successful return, a program should call `svsVideoGetImage` to retrieve the image data.

```
void svsVideoGetImage(unsigned char **left, unsigned char **right,
                      unsigned char **color)
```

Gets a stereo pair from the input capture buffers. `left` will contain a pointer to the left image array, and `right` a pointer to the right image array. No data is copied by this call. If the program intends to save this image data past the next invocation of `svsGetImage`, it should be copied, since the buffers will receive new data.`

The data format for the left and right images is 8 bits/pixel grayscale, packed as a linear array of size `width` x `height`.

If color information is available (the `color` parameter of the `svsSP` structure), it is returned in the `color` array; otherwise this array is set to NULL. In the color array, each pixel is represented as a set of four bytes, in the order RGBX.

5.4 Single Capture

In single capture mode, the program requests a single stereo pair, and then waits until it is delivered. The sequence of function calls is `svsVideoGrabSingle - svsVideoWaitImage – svsVideoGetImage`. As in continuous capture, the frame size and digitization parameters are downloaded to the camera hardware before the acquisition.

`int svsVideoGrabSingle(svsSP *sp)`

Requests a single stereo pair be captured. The video device(s) must already have been opened with `svsVideoOpen`. `svsVideoWaitSingle` can be used to check if the stereo pair is ready.

Returns 0 if successful, -1 if not. The biggest reason for an unsuccessful return (other than the camera not being opened) is that the frame size parameters are inconsistent. Use `svsVideoCheckParams` to verify the consistency of the parameters.

5.5 Stereo Engine functions

These functions compute a stereo disparity or confidence image from an input stereo pair. Stereo processing is affected by many parameters, such as the correlation window size, confidence filter threshold, etc. A parameter structure (`svsSP`) contains most of these; a few others are global variables.

5.5.1 Parameter Structure `svsSP`

This structure is used by most Stereo Engine functions. It holds image size and stereo processing parameters. A parameter structure can be saved to a file and reloaded from a file. Please see the definition in the file `src/svs.h`.

`svsSP svsParams`

This global variable contains an `svsSP` structure that is the default structure for most functions, if an explicit parameter structure is not passed as an argument.

5.5.2 MMX

This function and variable control the use of MMX instructions on Pentium processors. Normally, MMX is enabled.

`int svsHasMMX`

This global integer controls whether optimized MMX code is being used. User programs can set this flag to 0 (no MMX) or 1 to control whether such code is used. Normally, the SVS library will call `svsCheckMMX` to determine the correct state of svsHasMMX before any stereo processing.

`int svsCheckMMX(void)`

Checks whether the host computer supports MMX code, and sets the value of `svsHasMMX` appropriately. This check is done internally by the stereo function, and can also be called by user programs. Returns the value of `svsHasMMX`.
If `svsHasMMX` has already been set to 0 or 1, then `svsCheckMMX` does nothing. The value of `svsHasMMX` must be negative (unknown) for `svsCheckMMX` to have an effect.
Currently `svsCheckMMX` does not work under Windows 95/NT, and the default is to set `svsHasMMX` to 1.

5.5.3 Warping

Warping is used to correct for lens distortion, and to rotate the images to bring them into alignment as in a perfectly parallel stereo setup. It uses the distortion and external parameters in an `svsSP` structure. If desired, warping should be done before calling the stereo calculation functions.

```
void svsWarpImage(unsigned char *dest, unsigned char *src, int which,
                  svsSP *sp)
```

Warps the image pointed to by `src` into the buffer `dest`. This function corrects for radial distortion and rectifies the images to align the epipolar geometry. The warping uses an internal buffer, so `src` and `dest` can be the same buffer, if desired. If `sp` is `NULL`, the default parameter structure `svsParams` will be used.
The `which` parameter says which image is being rectified, left, right or left-color. Symbolic names for these are in `svs.h`:

`svsLEFT`

```
svsRIGHT
svsLEFTCOLOR
```

5.5.4 Stereo Calculation

These functions perform the stereo calculations. The `svsCalcStereo` function performs the full stereo calculation from intensity images, including pre-filtering, correlation, and post-filtering. It does not perform a warping step on the initial images to undistort them; to do this, first call the `svsWarpImage` function.

```
void svsCalcStereo(short *dest, unsigned char *il,
                   unsigned char *ir, svsSP *sp)
```

Compute disparity based on the images given in `il` and `ir`, and put the result into `dest`. Stereo parameters of the image are contained in `ps`. If `ps` is NULL, then the global variable `svsParams` is used.

To work with the result disparity image (`dest`), you have to understand how the disparity image is formatted. There are two issues: how disparities are encoded in pixels, and how disparity pixels correspond spatially to the original image.

Pixels in the disparity image are short integers (16 bits). For a valid disparity measurement, a pixel contains a value from 0 to 16`n`-1, where `n` is the number of disparities requested in `ps`. The factor of 16 appears because disparities are always interpolated to 1/16 pixel. For a typical disparity search range of 32, valid disparities are 0 to 511.

For pixels that are filtered by the texture measure or left/right check, two special values are used. 0xFFFF (-1) indicates a texture below the threshold, and 0xFFFE (-2) indicates a left/right check failure. All valid disparities are positive.

Because of windowing effects, the disparity image is smaller than the original intensity images. For consistency, the `dest` array is the same size as the input images. The disparity values are in the upper left corner of `dest`, using the same line width as the original image. The disparity image is spatially registered with the left intensity image. To find corresponding pixels, consult the following diagram.

The width, height, and offsets of the disparity image vary according to the stereo parameters in `sp`. The width and height of the disparity image are affected by the number of disparities, the size of the correlation window, and the external offsets of the images. The correct offsets of the disparity image are given by the `RESWIDTH`, `RESLEN`, `RESTOP`, and `RESLEFT` macros in `src/svs.h`. These macros take an `svsSP` pointer argument, the same one as given to the `svsCalcStereo` function.

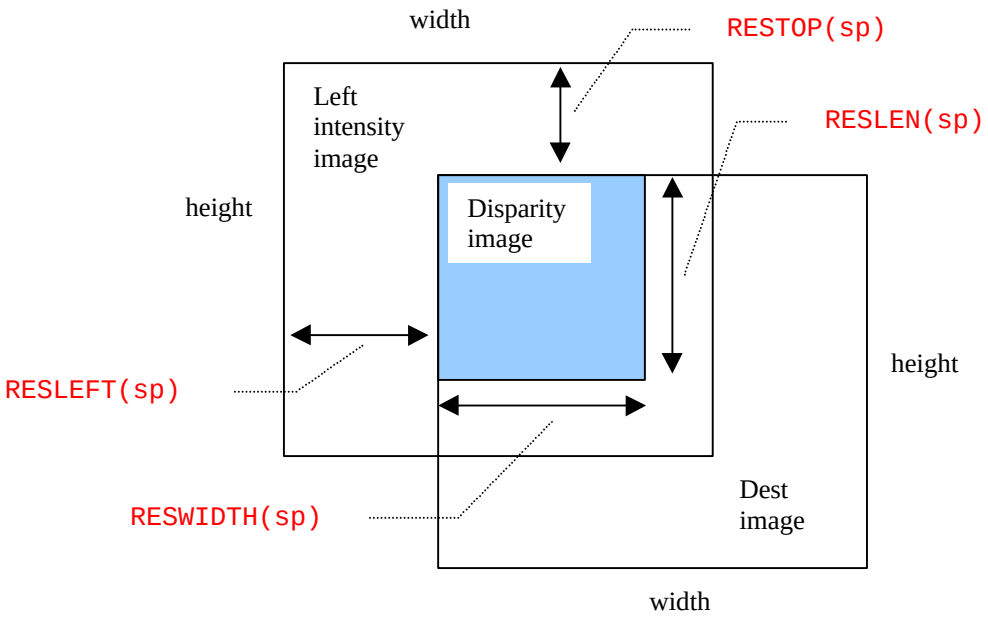


Figure 5-1 Spatial placement of the disparity image relative to the left input image.

```
void svsCalcConfidence(unsigned char *dest, unsigned char *il,
                       svsSP *sp)
```

Compute the confidence and return it in `dest`. See `svsCalcStereo` for a description of the various arguments.

5.5.5 3D Reconstruction

There are two functions for reconstructing points in 3D from (x, y) locations in the left image and their disparity values. The difference between the two is a trade off between accuracy and speed, with the first being more accurate but slower. All returned values are in millimeters.

```
void svsReconstruct3D(float *X, float *Y, float *Z,
                      float x, float y, float disp, svsSP *sp)
```

Reconstructs point (x, y, disp) in left rectified image, returning 3D point (X, Y, Z) in left camera coordinate system. Reconstruction is performed by least squares ray intersection, so it is slow but well behaved.

```
void svsReconstruct3DFast(float *X, float *Y, float *Z,
                          float x, float y, float disp, svsSP *sp)
```

Reconstructs point (x, y, disp) in left rectified image, returning 3D point (X, Y, Z) in left camera coordinate system. Reconstruction is fast, but sensitive to the stereo rig getting out of calibration.

5.6 Host display functions

These functions display images from the Stereo Engine in their proper format. As of SVS Version 2.0, display functions are derived from the FLTK cross-platform windowing system. All of the FLTK functionality is available in SVS, upon loading the FLTK shared library (`libfltk.so` or `fltkdll.dll`). Documentation on FLTK is available at www.fltk.org.

```
class svsWindow : public Fl_Window
```

Window class for display of input images or stereo results. Because svsWindow is a subclass of Fl_Window, any FLTK drawing function can be used on it. To superimpose graphics on the window, you would subclass the svsWindow, and define a new draw() member function that first called svsWindow::draw(), then your own drawing functions.

```
svsWindow::svsWindow(int x, int y, int h, int w)
```

Defines a new window for display. X and y are the coordinates or the upper left-hand corner of the window (relative to its parent window), and h and w are the height and width of the window.

```
void svsWindow::DrawImage(unsigned char *im, svsSP *sp,
                          int type = svsNONE)
```

Draws the image pointed to by `im` into the window. The parameter set `sp` defines the structure of the image. The draw function will fit the image into the window by decimation by powers of 2 in both vertical and horizontal dimensions, independently. The type svsNONE signals a standard 8-bit grayscale image; the type svsSTEREO is for disparity images. Since disparity images are short ints, they must be cast to unsigned char.

`double sp.gamma`

The photosites in most cameras have a linear response to light, so the pixel value is a linear function of the amount of light the photosite receives. However, a monitor displaying this value normally has a nonlinear output, so that darker values are perceived as too dark. Most cameras compensate by applying a nonlinear function, x^g, which boosts the low-light values of the video stream so they display properly. But some cameras, in particular the MEGA-D, return the uncompensated linear signal. To display the video stream from these cameras, the `DrawImage` function can apply a gamma correction. The value of `gamma` in the `sp` parameter structure transforms the linear pixel values to their gamma-corrected values for drawing. A value of 0.0 or 1.0 means no gamma correction; values between 0.0 and 1.0 apply the corresponding correction. The standard correction is 0.45.

```
void ClearImage()
```

Clears any image in the window and replaces it with black.

5.7 Host file functions

```
unsigned char *svsReadFile(char *name, unsigned char *buffer,
                           int *height, int *width, int *num)
```

This function reads from the SSI file given by the string name into the unsigned character buffer specified in `buffer`. If `buffer` is NULL, then a new one will be created, and is the return value of the function. The arguments `height`, `width`, and `num` are all pointers to integers where the size of the images, along with the number of stereo pairs will be stored.

```
unsigned char *svsReadFile2BMP(char *name, unsigned char *buffer,
                               int *height, int *width)
```

This function reads from the two BMP files given by the string name into the unsigned character buffer specified in `buffer`. If `buffer` is NULL, then a new one will be created, and is the return value of the function. The two images are stored consecutively in the buffer. The arguments `height` and `width` are pointers to integers where the size of the images will be stored.

By convention, stereo pairs are stored as two BMP files, with names XXX-L.bmp and XXX-R.bmp. svsReadFile2BMP will accept either the base name XXX or either full name, and retrieve both files.

```
int svsReadFileBMP(char *name, unsigned char *buffer,
                   int *height, int *width)
```

This function reads from the BMP file given by the string name into the unsigned character buffer specified in `buffer`. If `buffer` is NULL, then a new one will be created, and is the return value of the function. The arguments `height` and `width` are pointers to integers where the size of the images will be stored.

```
int svsReadParamFile(char *fname, svsSP *sp)
```

Reads in a parameter file with path `fname` and stores the values in the structure `sp`. If `sp` is `NULL`, the values are stored in the global `svsParams`. For a description of the parameter file format, see Section 2.6.

```
int svsWriteParamFile(char *fname, svsSP *sp)
```

Saves the parameter settings in sp into the file with path `fname`.